UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

City Challenge

propusă de

Iulian Alexandru Vlad

Sesiunea: *Iulie, 2015*

Coordonator științific

Asist. Dr. Vasile Alaiba

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI
FACULTATEA DE INFORMATICĂ

City Challenge

Iulian Alexandru Vlad

Sesiunea: *Iulie, 2015*

Coordonator științific
Asist. Dr. Vasile Alaiba

DECLARAȚIE PRIVIND ORIGINALITATE ȘI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul „City Challenge” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imaginile etc. preluate din proiecte open-source sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași, *25.06.2015*

Absolvent *Iulian Alexandru Vlad*

_____

(semnătura în original)

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „City Challenge”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” Iași să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *25.06.2015*

Absolvent *Iulian Alexandru Vlad*

_____

(semnătura în original)

Cuprins

Introducere

Context

Orientarea sportivă datează încă din anul 1850, fiind organizată de mediile militare scandinave ca mijloc de pregătire. Primul astfel de concurs, numit ”Concurs sportiv de Orientare”, a avut loc în Bergen, Norvegia la 13 mai 1897.

Orientarea este o familie de sporturi unde concurenții au ca scop parcurgerea unui traseu necunoscut utilizând doar o hartă și o busolă. De asemenea, orientarea presupune îmbinarea aptitudinilor fizice și intelectuale. Premierea jucătorilor se stabilește în funcție de timpul pe care concurentul l-a realizat pentru parcurgerea traseului și a posturilor de control, în ordinea stabilită; câștigătorul este cel care are cel mai mic timp. Sportul de orientare este împărțit în mai multe categorii, precum: orientarea în alergare, orientarea utilizând schiurile, orientarea pentru persoane cu handicap, orientarea călare și orientarea utilizând bicicleta[1].

Principala categorie este reprezentată de orientarea în alergare, care presupune parcurgerea unui traseu necunoscut constituit dintr-o succesiune de posturi de control. Instrumentele pe care concurenții le au la dispoziție sunt busola și o hartă, care include detalii referitoare la traseul pe care trebuie să îl parcurgă jucătorii, precum: vegetația și relieful.

Motivație

De-a lungul timpului, cele mai cunoscute concursuri de orientare turistică se bazau pe instrumente precum: hartă, busolă sau hartă și busolă. În funcție de tipul de concurs la care participă concurentul, diferă și instrumentele pe care participantul le primește. Spre exemplu, într-un concurs ce presupune doar folosirea hărții, persoanele care participă la activitate primesc o hartă specială a unei zone bine delimitate, pe care sunt marcate toate punctele de control. Folosind harta, concurenții trebuie să descopere poziția punctelor de control și să reușească să le atingă pe toate în cel mai scurt timp.

[1] ***, Orientarea turistică – Fișă tehnică, Disponibil la: http://www.scoutpanaitescu.ro/campuri-activitati/Orientare_turistica.pdf

Odată cu evoluția tehnologiei, a apărut ideea de *GPS (Global Positioning System)* care este un sistem global de navigație folosind sateliți si unde radio. Ca și principiu de funcționare, GPS-ul utilizează sateliții din spațiu ca puncte de referință pentru localizarea la sol. Primul telefon care a încorporat acest sistem a fost Nokia și a folosit Glonass, care era o replică la Global Positioning System din SUA[2].

În prezent, tot mai multe persoane folosesc diferite dispozitive precum tablete, laptop-uri, telefoane și toate acestea au încorporate sisteme de GPS. Utilizarea unui astfel de dispozitiv drept instrument într-un concurs de orientare turistică ar putea să capteze atenția cât mai multor participanți și să incite la competiție.

Obiectiv

Îmi propun realizarea unei aplicații care să îmbine cele două concepte menționate anterior și mai mult decât atât, să ofere utilizatorilor noi provocări nu doar găsirea punctelor de control într-un timp cât mai scurt. Pentru aceasta, aplicația va pune la dispoziția utilizatorului trei moduri de joc prin care se testează atât abilitățile participantului la joc de a găsi punctele de control într-un timp cat mai scurt, cât și atenția acestuia la detaliile din jurul său prin rezolvarea unei ghicitori la fiecare post de control.

Structura lucrării

Lucrarea de față cuprinde trei capitole și anume: *Platforma de dezvoltare Android*, *Tehnologii folosite* și *Arhitectura aplicației și detalii de implementare*. Prin intermediul acestora îmi propun să înfățișez necesitatea unei astfel de aplicații în contextul evoluției tehnologiei, prezentarea celor mai favorabile tehnologii în dezvoltarea acestei aplicații și descrierea în detaliu a utilizării acestora și în cele din urmă, expunerea amănunțită a dezvoltării jocului.

Contribuții personale

Am realizat o aplicație care oferă un cadru de joc atractiv și distractiv, și care să atingă obiectivele propuse și menționate anterior. Prin urmare, am realizat o aplicație care are trei moduri de joc după cum urmează: în modul *singleplayer*, utilizatorul trebuie să completeze un traseu într-un timp cât mai scurt, iar pentru a trece de la un punct de control la altul, trebuie să rezolve un

[2]***, Nokia, primul telefon din lume cu GPS rusesc, Disponibil la: http://www.money.ro/nokia-primul-telefon-din-lume-cu-gps-rusesc/

puzzle la fiecare punct la care a ajuns. Odată ce a oferit răspunsul corect la întrebare, i se va sugera un posibil drum către următorul punct de control, însă jucătorul poate selecta traseul favorit, doar să ajungă cât mai repede la final. Dacă utilizatorul alege modul *multiplayer,* jocul se va desfășura în mod asemănător, singura diferență fiind că acesta va primi în timp real notificări cu privire la stadiul în care se află adversarul și chiar are posibilitatea de a-i dezvălui locația de trei ori. În cel de-al treilea mod, modul creativ, utilizatorul poate crea o anumită hartă pentru a juca cu un alt participant, sau pentru a o face publică celorlalți utilizatori ai aplicației.

La nivel arhitectural, aplicația este formată dintr-un client implementat pe platforma Android și un server, realizat folosind limbajul de programare C++, pentru toate procesările necesare clientului. De asemenea, pentru stocarea datelor am folosit biblioteca SQLite pentru C++, iar legătura dintre client și server se realizează prin comunicarea pe socket-uri utilizând protocolul TCP.

Capitolul I. Platforma de dezvoltare Android

În prezent, Android este una din cele mai renumite platforme de dezvoltare oferind posibilitatea realizării de aplicații pentru milioane de dispozitive din întreaga lume.

Rolul dezvoltatorilor de aplicații mobile s-a schimbat dramatic în ultimii anii și au ajuns să fie foarte căutați pe piața de muncă. Producătorii de telefoane, furnizorii de platforme și chiar și operatorii de rețea se află în competiție pentru realizarea celei mai mari comunități de dezvoltare.

Conform statisticilor (*Fig.1.1*), Android deține monopolul în ceea ce privește platforma de dezvoltare pentru aplicații mobile urmat îndeaproape de iOS.

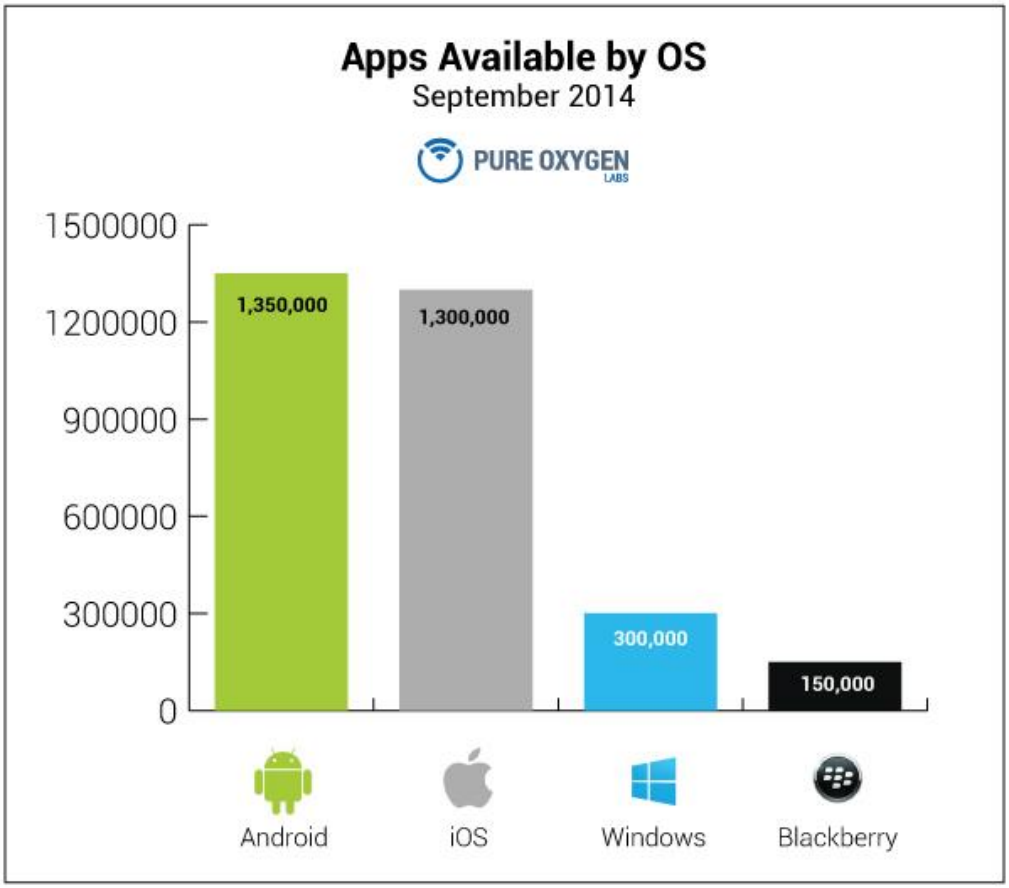

Fig. 1.1 Statistici privind dezvoltarea de aplicații mobile[3]

[3] Jay P., September 2014 statistics about all app store, Disponibil la: http://www.whatech.com/mobile-apps/news/32469-september-2014-statistics-about-all-app-store

1.1 SDK-ul Android

SDK-ul Android conține un set de instrumente de dezvoltare, precum biblioteci, program de depanare, un emulator de dispozitiv, documentație, mostre de cod și tutoriale menite să simplifice dezvoltarea de aplicații mobile. Platformele de dezvoltare sprijinite în prezent înglobează calculatoare pe X86 care rulează Linux, Mac OS sau mai recent Windows 7 sau 8.1.[4]

Îmbunătățirile aduse la SDK-ul Android facilitează dezvoltarea de aplicații destinate primelor versiunii ale platformei Android, astfel se pot realiza aplicații și pentru dispozitivele mai vechi. Instrumentele pentru dezvoltare pot fi descărcate în funcției de platforma și versiunea dorită pentru a se asigura compatibilitatea acestora cu dispozitivele pentru care sunt realizate.

Mediul de dezvoltare este *Android Studio* lansat în 2013 cu scopul de a facilita dezvoltarea de aplicații pentru persoanele care nu dețin cunoștințe avansate de programare, promițând realizarea de aplicații mobile într-un mod simplu și intuitiv. Android Studio oferă o serie de *template-uri* în funcție de necesitatea aplicației ce simplifică semnificativ realizarea de aplicații. De asemenea, furnizează module care permit dezvoltarea rapidă de aplicații de captură, sisteme de notificări sau platforme de mesagerie instant. O altă facilitate este reprezentată prin varietatea foarte mare de dispozitive și dimensiuni de ecran, fapt ce face ca Android Studio să fie un modul ideal pentru realizarea de aplicații mobile care să fie compatibile pe toate dispozitivele.

Acest suport pentru aplicații are un set de servicii și sisteme precum: extensii de vizualizări (Views), furnizorii de conținut (Content Providers), manager de resurse (Resource Manager), managerul de notificări (Notification Manager) și activități (Activity Manager).

Extensiile de vizualizări pot fi utilizate la dezvoltarea de aplicații prin utilizarea de liste (ListView), care permit afișarea conținutului într-un mod elegant permițând opțiunea de scroll; alte extensii de vizualizări sunt: gridurile, casetele de text, butoanele și browserele web.

Un alt serviciu se referă la furnizorii de conținut, care permit accesul la date din alte aplicații, ca și Contacts sau partajarea propriilor date, dar și realizarea de mecanisme pentru asigurarea securității datelor.

[4] Stanciu L., Programare vizuală și modelare, Disponibil la: http://www.aut.upt.ro/~loredanau/teaching/PVM/Programare%20Vizuala%20si%20Modelare_intermediar.pdf

Managerul de resurse oferă accesul la resursele care nu sunt cod și anume fișiere, grafice și altele, iar managerul de notificări permite afișarea de notificări pe bara de stare.

Și în cele din urmă, *managerul de activități* (Activity Manager), care are rolul de a gestiona ciclul de viață al aplicațiilor, precum și navigarea comună prin introducerea unei bare de navigare cu un buton pentru opțiunea de revenire la paginile anterioare.[5]

1.2 Aplicații populare

Într-o eră digitală, în care toata lumea folosește din ce in ce mai mult dispozitivele electronice, o aplicație care să realizeze conexiunea dintre realitate si virtual ar trezi interesul multor persoane care folosesc aceste dispozitive.

În continuare, vom prezenta pe scurt câteva din aplicațiile populare ale momentului pentru sistemele de operare Android care combină lumea reală cu tehnologia.

1.2.1 Ingress

Ingress este un joc multiplayer masiv creat de Google și Niantic labs și lansat in Decembrie 2014[6].

Participanții își copie aplicația apoi își folosesc telefonul pentru a se autentifica la interfața bazată pe Google Maps. Odată intrați în joc, utilizatorii pot vedea portale în jurul lumii. Aceste portale sunt amplasate la locații culturale semnificative precum monumente, muzee. În joc există doua facțiuni numite Iluminații și Rezistența care se află în război pentru aceste portale. Când un participant la joc se află lângă un portal, poate prelua acel portal pentru echipa sa, poate seta anumite elemente defensive la acea locație sau poate stabili legături cu celelalte portale din teritoriul echipei sale conform *Fig. 1.2.*

[5] ***, Prezentare Android, Disponibil la: http://ocw.cs.pub.ro/courses/pdsd/labs/00

[6] ***, Ingress: The game that reveals Google's secret war to control London, Disponibil la: http://www.theguardian.com/technology/2014/jun/04/ingress-the-game-that-reveals-googles-secret-war-to-control-london

Fig. 1.2 Ingress București

1.2.2 Parallel Kingdom

Parallel Kingdom este un joc care plasează utilizatorii într-un univers paralel prin transformarea Google Maps într-o imensa harta cu regate, peșteri si castele[7], conform *Fig. 1.3.*

Ideea jocului este destul de simplă. Jucătorii trebuie să omoare monștrii din peșterile întunecate sau să se lupte cu dragonii din propria grădină. Participanții la joc pot călători pe hartă în căutarea de noi tărâmuri pentru a le cuceri. În timpul acestor călătorii, jucătorii pot colecționa diferite unelte, mâncare, își pot îmbunătăți abilitățile și se pot coordona cu ceilalți aliați datorita sistemului de conectivitate pe care îl are implementat jocul.

Acest joc se bazează în principal pe servicii de localizare și implementează conceptul de realitate augmentată.

[7] Nuñal P., Best augmented reality (AR) games for Android, Disponibil la: http://www.androidauthority.com/best-augmented-reality-ar-games-for-android-93520/

Fig. 1.3 Captură în diverse ipostaze ale jocului incluzând meniul

1.2.3 Zombie, Run

Zombie, Run este un joc în care, de asemenea, se utilizează ideea de realitate augmentă combinată cu sistemele de localizare ale dispozitivului electronic[8].

În această aplicație, se transforma Google Maps într-un tărâm plin cu zombie care bântuie în căutarea de oameni vii. Scopul jucătorului este de a fugi de acești zombie pentru a-și salva viața. Acesta își poate selecta nivelul de dificultate prin selectarea numărului de zombie care să apară pe hartă si ce viteză de deplasare au aceștia conform *Fig. 1.4*. Odată cu începerea jocului, utilizatorul trebuie să alerge până la un anumit punct de final fără a intra în contact cu nici un zombie.

[8] Saket, 36 Awesome Augmented Reality Apps & Games for Android, Disponibil la: http://techsplurge.com/3214/mega-list-33-awesome-augmented-reality-apps-games-android/

Fig. 1.4 Imagini din timpul jocului

1.2.4 SpecTrek

SpecTrek este o aplicație care utilizează locația utilizatorului și camera dispozitivului pe care este instalată aplicația pentru a simula o vânătoare de fantome[9].

Aplicația detectează locația jucătorului pentru a plasa fantomele în apropierea acestuia. Obiectivul jocului este de a găsi aceste fantome și să le captureze. Pentru a captura fantomele trebuie ca utilizatorul să se apropie cât mai mult de acestea pentru a arunca peste ele o plasă specială conform *Fig. 1.5*.

[9] ***, Top 10 Augmented Reality Games for Android / iOS platforms, Disponibil la: http://deepknowhow.com/2013/12/11/augmented-reality-games-android-iphone/

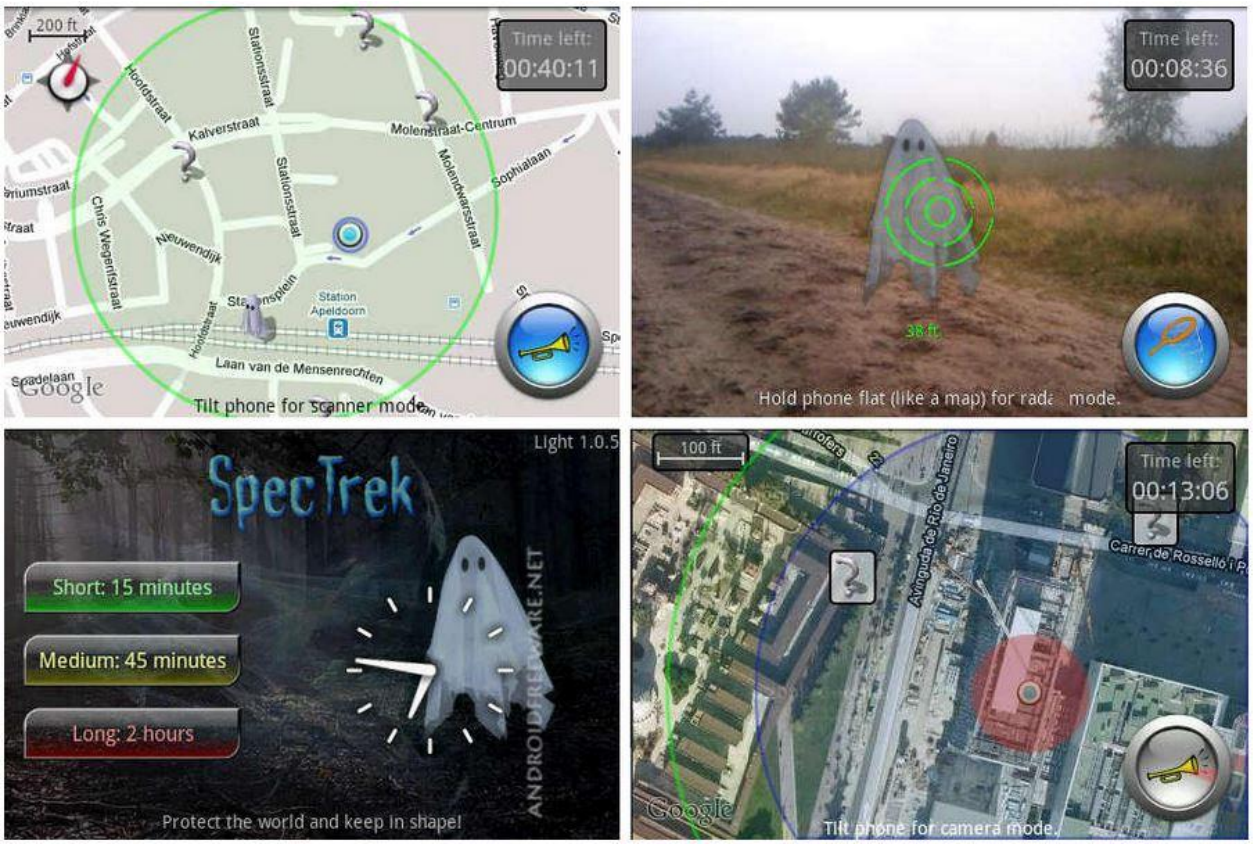

Fig. 1.5 Capturi din aplicație

1.3 Concluzii

În prezent, datorită numărului de persoane în continuă creștere care folosesc dispozitive mobile, dezvoltarea de aplicații pentru aceste dispozitive se află într-un progres permanent. Astfel, zilnic apar zeci de aplicații de acest fel, unele mai reușite iar altele mai puțin reușite. Ceea ce ar face un joc interesant și să incite utilizatori la participare este îmbinarea realității cu mediul virtual oferit de telefon sau tabletă.

Astfel, am decis să implementez un joc care să realizeze aceasta îmbinare și să aibă ca și țintă sistemul de operare Android, fiind unul din cele mai populare sisteme de operare și oferind cea mai mare si performantă platformă de dezvoltare pentru aplicații.

Capitolul II. Tehnologii folosite

Pentru dezvoltarea jocului ”*City Challenge*” au fost folosite următoarele tehnologii:

- Sistemul de operare Android
- Google Maps API
- Server C++
- Protocolul TCP
- Socket-uri BSD
- Fire de execuție
- SQLite

2.1 Sistemul de operare Android

Android este un sistem de operare pentru dispozitive și telefoane mobile bazată pe nucleul Linux, dezvoltată inițial de compania Google, iar mai târziu de consorțiul comercial Open Handset Alliance. Android permite dezvoltatorilor să scrie cod gestionat în limbajul Java, controlând dispozitivul prin intermediul bibliotecilor Java dezvoltate de Google. Aplicații scrise în C și în alte limbaje pot fi compilate în cod mașină ARM și executate, dar acest model de dezvoltare nu este sprijinit oficial de către Google. Lansarea platformei Android, la 5 noiembrie 2007, a fost anunțată prin fondarea Open Handset Alliance, un consorțiu de companii de hardware, software și de telecomunicații, consacrat dezvoltării de standarde deschise pentru dispozitive mobile. Google a lansat cea mai mare parte a codului Android sub licența Apache, o licență de tip free-software si open-source.

Începând cu 21 octombrie 2008, Android a fost disponibil ca *Open Source*. Google a deschis întregul cod sursă, care anterior era disponibil sub licența Apache. Aceasta permite producătorilor să adauge extensii proprietare, fără a le face disponibile comunității open source. În timp ce contribuțiile Google la această platformă se așteaptă să rămână open source, numărul versiunilor derivate ar putea exploda, folosind o varietate de licențe.

2.1.1 Arhitectura sistemului de operare: Android

În continuare aș dori să prezint arhitectura *sistemului de operare Android* care cuprinde cinci secțiuni pe patru nivele, acestea fiind: Kernelul Linux, biblioteci, motorul Android și cadrul pentru aplicații.[10]

Kernelul Linux utilizează o serie de patch-uri pentru versiune oficială, și conține toate driverele pentru componentele hardware, precum camera, tastatura, WiFi și dispozitive audio. De asemenea, la Kernelul Linux au fost adăugate unele funcționalități specifice Android, acestea fiind: wakelocks, low-memory killer, binder, logger.

Wakelocks este o soluție pentru problema de power management folosită de Android, care are ca scop reducerea consumului prin intrarea în starea de ”sleep” pentru momentele în care nu este utilizat.

În ceea ce privește *low memory killer*, scopul său este de opri componentele care nu au fost folosite o perioadă lungă de timp.

Următoarea funcționalitate este reprezentată de *binder*, un mecanism de RPC/IPC, care are ca scop capabilitate de invocare remote a obiectelor asemănătoare cu obiectele COM din Windows.[11]

Și în cele din urmă *logger*, care are ca scop substituirea *sistemului clasic de logging* al kernelului, cu scopul diminuării numărului de task switch-uri și scrieri în fișier, printr-un buffer circular.

O altă componentă este *motorul Android*, alcătuit dintr-o serie de biblioteci de bază, care permit utilizatorilor să dezvolte aplicații mobile, folosind ca limbaj de programare Java; aceste biblioteci permit accesul la funcțiile unui dispozitiv și anume: telefonie, mesaje, gestiunea pachetelor. La baza arhitecturii stau regiștri, fiind echipată cu un compilator JIT, care permite modificarea executabilului obținut pe dispozitivul mobil.

Următoarea componentă este cadrul pentru aplicații: *Android framework*, care furnizează diverse funcționalități ale sistemului de operare pentru ca programatorii să le poată transpune în aplicațiile lor. Această componentă oferă dezvoltatorilor posibilitatea de a realiza aplicații

[10] ***, Prezentare Android, Disponibil la: http://ocw.cs.pub.ro/courses/pdsd/labs/00
[11] ***, Android, Disponibil la: http://cs.curs.pub.ro/wiki/si/lab/2013/android

complexe și inovative, întrucât aceștia sunt liberi să utilizeze hardware-ul echipamentelor, de informațiile despre locație, rularea de servicii în background, setarea de alarme, precum și adăugarea de notificări pe bara de stare. Programatorilor li se oferă acces la aceleași API-uri ca și aplicațiile distribuite cu Android, prin urmare arhitectura aplicațiilor este proiectată astfel încât să fie simplificată reutilizarea componentelor: o aplicație poate publica anumite funcționalități, și o altă aplicație să le poată utiliza.

Ultima componentă este reprezentată de *nivelul aplicații*, care oferă atât produsele încorporate în dispozitivele mobile, precum: Camera, Music player, Contacts, și Video player, cât și produsele disponibile pe Play Store.

2.1.2 Funcționalitățile sistemului de operare Android

Principalele funcționalități pe care sistemul de operare Android le oferă sunt: *stocare*, care folosește SQLite, bază de date relațională ce permite utilizarea eficientă a resurselor; *conectivitatea* prin diverse modalități, precum: 3G, WiFi, Bluetooth, WiMAX, GPRS, EDGE; *WiFi direct*, care permite interconectarea între diverse dispozitive având o lățime de bandă mare; *Android Beam*, prin care utilizatorii partajează conținut instant prin apropierea dispozitivelor respective. O altă funcționalitate este *navigarea pe Internet* bazată pe motorul open source pentru navigare WebKit împreună cu motorul JavaScript de la Chrome V8 suportând HTML5 și CSS3.[12]

Multimedia admite mai multe formate precum: H.263, M-peg-4, AMR-WEB, AAC, JPEG; *multi-touch*, care suportă posibilitatea de contact în mai multe puncte concomitent; *multi-tasking*; *GCM*(Google Cloud Messaging) permițând dezvoltatorilor expedierea de date de dimensiuni reduse, în lipsa unei soluții de sincronizare proprietară.

2.2 Google Maps API

Google a lansat în anul 2005 Google Maps API[13], care permite programatorilor integrarea de hărților de la Google în propriile aplicații; acest serviciu fiind gratuit. API-ul permite accesul la serverele Google Maps, descărcarea de date, afișarea unei hărți, și trimiterea unui răspuns la interacțiunea cu harta.

[12] ***, Prezentare Android, Disponibil la: http://ocw.cs.pub.ro/courses/pdsd/labs/00
[13] ***, Google Maps Android API ,Disponibil la: https://developers.google.com/maps/documentation/android/map

O altă facilitate pe care API-ul Google Maps o furnizează este inserarea de informații suplimentare despre obiectele de pe o porțiune a hărții, astfel încât să permită interacțiunea utilizatorului cu harta. Totodată, acesta permite adăugarea de obiecte grafice pe hartă, precum: ancore pentru pozițiile specifice pe hartă, cunoscute sub denumirea de markeri, segmente de linii, segmente închise, imagini și elemente grafice de tip bitmap atașate la poziții specifice pe hartă.

Google Maps API Android v2 include un utilitar pentru accesibilitate numit TalkBack. În momentul în care acesta este pornit, orice glisare a utilizatorului pe ecran muta focalizarea de la un element grafic la altul. Când un element grafic este focalizat, utilitarul TalkBack citește numele acestui element. Dacă utilizatorul apasă de două ori pe ecran, acțiunea focalizată este realizată.

Hărțile afișate de Google Maps API prezintă următoarele caracteristici: titlul acestora nu includ conținut personalizat, iar referitor la pictograme, nu toate permit acțiunea de click. În plus față de funcționalitea de cartografie, API sprijină o gamă completă de interacțiuni cu harta, în concordanță cu modelul Android UI, un exemplu ilustrativ este posibilitatea configurării interacțiunilor cu harta, prin definirea de *listeneri*, concept care are ca scop oferirea de răspunsuri la gesturile utilizatorilor. Principala clasă care se utilizează în cazul utilizării hărților este *GoogleMap*. Aceasta modelează harta în cadrul aplicației, iar în cadrul interfeței utilizator, harta va reprezentată printr-un MapFragment sau MapView. Clasa *GoogleMap* permite realizarea următoarelor acțiuni: conectarea la serviciul Google Maps, descărcarea componentelor hărții, afișarea de diverse controale, precum zoom și răspunderea la acțiunea de zoom.

MapFragment este o subclasă Android, care permite plasarea unei hărți într-un fragment Android și oferă acces la obiecte de tip GoogleMap. Spre deosebire de o extensie de vizualizare (View), un fragment reprezintă un comportament sau o porțiune din interfața utilizator dintr-o activitate, astfel o activitate poate include mai multe fragmente. De asemenea, un fragment poate fi utilizat în mai multe activități, dar și utilizarea mai multor fragmente pentru realizarea unui interfețe.

În ceea ce privește *MapView* este o subclasă a clasei *Android View*, care permite plasarea hărții într-o extensie de vizualizare. Un View reprezintă o regiune a ecranului, element esențial în dezvoltarea de aplicații Android și widget-uri. Mai mult decât atât, prin intermediul unui MapFragment, MapView-ul acționează asemănător unui container pentru hartă

2.3 Server C++

Cele mai multe comunicării între procese folosesc modelul client-server. Acest model presupune ca unul din cele doua procese numit *client,* se conectează la celălalt proces numit server pentru a îi cere o anumită informație. Este important faptul că procesul client trebuie să știe de existența și adresa server-ului pe când serverul nu trebuie să știe adresa clientului apriori stabilirii conexiuni între cei doi.

C++ oferă posibilitatea de a scrie cod foarte eficient și în același timp are un nivel de abstractizare ridicat, prin urmare, pentru a dezvolta o aplicație performantă, care sa ruleze pe orice tip de dispozitiv, un server scris în limbajul de programare C++ ar putea fi cea mai bună soluție.

2.4 Protocolul TCP

TCP (Transmission Control Protocol) este un protocol orientat pe conexiune ce urmărește oferirea calității maxime a serviciilor. Oferă conexiuni de încredere, bidirecționale, conexiunea TCP fiind un flux de octeți si nu un flux de mesaje.

Stabilirea conexiunii într-o comunicare TCP se realizează folosindu-se mecanismul de *Three-way handshaking.* Acest mecanism presupune trimiterea unui pachet inițial apoi se așteaptă confirmarea de primire de la destinatar în vederea trimiterii următorului pachet. Dacă confirmarea nu este primită într-un anumit interval de timp prestabilit, se retransmite pachetul, aspect redat în *Fig. 2.1.*

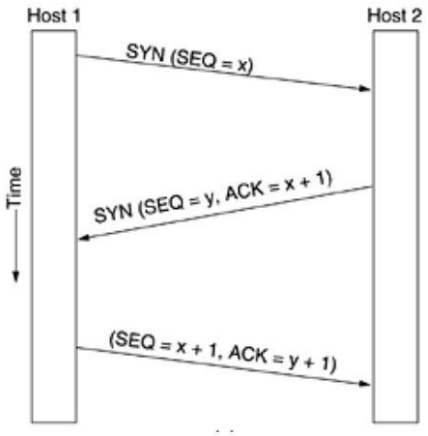

Fig. 2.1 Mecanismul Three-way handshaking[14]

[14] Andrew S. Tanenbaum, Computer Networks, 2010

Pentru comunicare, TCP folosește conceptul de fereastră glisantă ce presupune împărțirea fluxului de octeți în segmente. Segmentele trimise și confirmările vor transporta numere de secvență indicând poziția octeților transmiși în cadrul fluxului de date. Segmentele pierdute sunt detectate prin utilizarea unui timer de retransmisie a datelor iar pentru detectarea erorilor se utilizează sume de control.

Datorită conexiunii full-duplex, în momentul în care o aplicație nu mai are date de trimis, TCP va închide conexiunea doar într-o singură direcție. Conexiunea este înlăturată în momentul în care ambele direcții au fost oprite.

2.5 Socket-uri BSD

Cea mai populară metodă de programare utilizând protocolul TCP este prin utilizarea interfeței pentru socket-uri BSD (Berkeley System Distribution). Utilizând aceasta interfață, endpoint-urile rețelei sunt reprezentate ca și socket-uri. Conceptul de socket este independent de arhitectura hardware, de protocol si de tipul de transmisie a datelor. Este utilizat pentru comunicarea în rețea între procese aflate pe mașini diferite.

La nivel de implementare, este similar fișierelor, pipe-urilor, utilizându-se interfața de programare I/O existentă. Practic este identic cu un descriptor de fișier, singurele diferențe fiind la creare si câteva opțiuni de control al socket-urilor.

Pentru crearea unui server folosind interfața BSD trebuie realizații următorii pași[15]:

- Crearea unui nou socket folosind funcția *socket().*
- Atașarea unei adrese la socket prin funcția *bind().*
- Așteptarea unei noi conexiuni la socket prin utilizarea metodei *listen().*
- Acceptarea conexiuni noi cu ajutorul funcției *accept().*

După acceptarea conexiuni se realizează transmiterea de mesaje între server si client prin intermediul metodelor de *write(), read()* sau *recv(), send(),* până la închiderea conexiunii, aspect exemplificat în *Fig. 2.2.*

La nivel de client, implementarea este destul de simplă. Acesta trebuie doar sa creeze un socket, să apeleze metoda *connect()* pentru conectarea la server urmată apoi de trimiterea și

[15] Griffin I., Linux Network Programming, Part 1, Disponibil la: http://www.linuxjournal.com/article/2333

primirea de mesaje către și de la server, fapt ilustrat în *Fig. 2.2*. Apelarea metodei *bind()* pentru atașarea unei adrese este opțională.

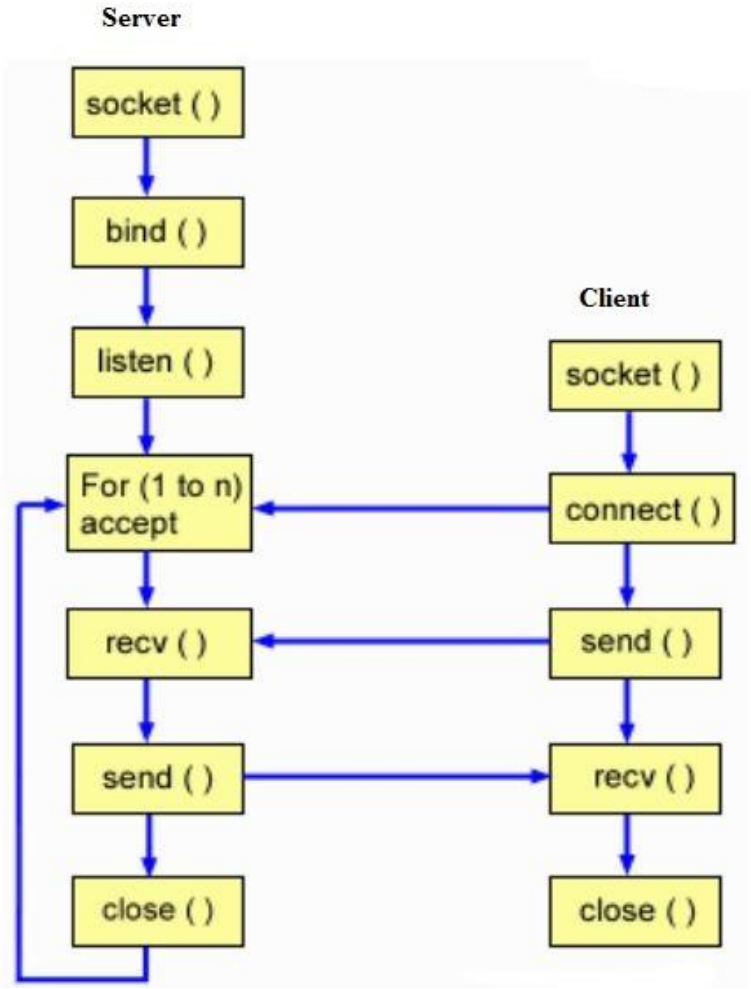

Fig. 2.2 Implementare folosind socket-uri BSD[16].

2.6 Fire de execuție

Un fir de execuție, numit și *thread,* este o secvență de instrucțiuni care poate fi executată concurent cu o altă secvență, într-un mediu *multithreading,* în timp ce partajează aceeași zonă de memorie. Acest fire de execuție rulează în paralel pentru a crește eficiența programelor. În sistemele cu procesoare multiple sau cu nuclee multiple, firele de execuție rulează în același timp

[16] ***, Diagramă disponibilă la: http://publib.boulder.ibm.com/

pe procesoare sau nuclee diferite. Pentru procesoarele cu un singur nucleu, sistemul împarte timpul de execuție între *thread*-uri, aspect prezentat în *Fig. 2.3*.

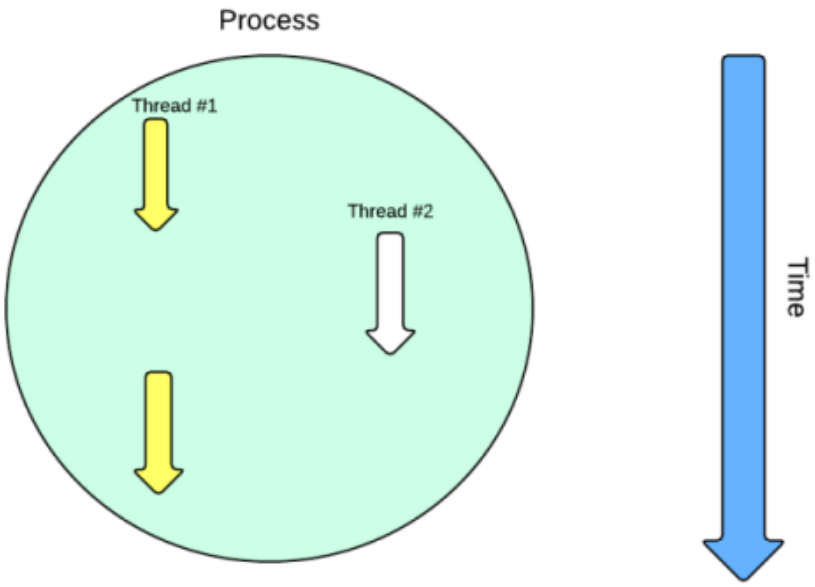

Fig. 2.3 Împărțirea timpului între firele de execuție[17]

La nivel de implementare se poate utiliza standardul *Pthreads (POSIX Threads)*, ce definește un API pentru crearea și manipularea firelor de execuție. Pentru implementarea unui server TCP care deservește clienți multiplii se utilizează acest standard deoarece aplicațiile ce folosesc fire de execuție sunt în general mai rapide decât dacă utilizează procese.

2.7 SQLite

SQLite este o bibliotecă care implementează o bază de date SQL tranzacțională și care nu are nevoie de o configurație și nici de un server. Codul pentru această librărie este public și poate fi folosit pentru orice scop.

Spre deosebire de alte baze de date SQL, SQLite nu are un server separat pentru procesare. Acesta citește și scrie direct în memoria sistemului. Oferă o bază de date complexă cu posibilitatea de a crea tabele multiple, indici, *trigger*-e și *view*-uri iar tranzacțiile sunt *ACID* (*Atomicity*,

[17] ***, Multithreading in C++, Disponibil la: https://www.tutorialcup.com/cplusplus/multithreading.htm

Consistency, Isolation, Durability) chiar dacă sunt întrerupte de erori. De asemenea, formatul fișierelor scrise de aceasta bibliotecă este *cross-platform* oferind posibilitatea de a fi utilizat atât pe sisteme de 32 de biți cât și pe cele de 64 sau pe arhitecturi de tip *big-endian* sau *little-endian*.

Datorită acestor facilitați, SQLite este opțiunea ideală în implementarea unei aplicații performante.

2.8 Concluzii

Întrucât intenționăm dezvoltarea unei aplicații cât mai performante, care sa ruleze cât mai eficient, pe orice tip de dispozitiv chiar și mai puțin performant, am ales tehnologiile mai sus menționate datorită avantajelor pe care acestea le prezintă.

Astfel, la nivel de server am utilizat C++ datorită performanțelor pe care le oferă acest limbaj prin managementul cât mai eficient a memoriei folosite și SQLite ca și suport de stocare a datelor deoarece este o bibliotecă simplă de utilizat și eficace în ceea ce privește memoria consumată. Totodată, pentru comunicare am ales protocolul TCP datorită siguranței pe care acesta o oferă referitor la transferul de date dintre clienți și server și datorită conexiuni stabile pe care acesta o stabilește între participanți. Pentru deservirea clienților multipli am ales comunicarea prin socket-uri grație conexiuni persistente care se realizează între client si server. De asemenea, această comunicare pe socket-uri folosind protocolul TCP este mai rapidă și mai stabilă decât, spre exemplu, comunicarea prin cereri HTTP.

La nivel de client, am ales sistemul de operare Android datorită popularității acestuia în rândul dispozitivelor digitale și pentru platforma de dezvoltare pe care acesta o oferă. Totodată, pentru realizarea concursului de orientare sportivă atât în modul singleplayer cât și multiplayer în care pot participa mai mulți jucători am utilizat API-ul Google Maps, pus la dispoziție de Google deoarece oferă o hartă ce poate fi modificată și pe care se pot insera diferite *marker*-e ce afișează punctele de control sau poziția adversarului. În același timp, pentru localizarea permanentă a jucătorului am utilizat sistemul de localizare oferit de Android pentru faptul ca oferă mereu indicații cu privire la schimbarea locației curente sau date legate de semnalul GPS prin implementarea conceptului de *listeneri.*

Capitolul III. Arhitectura aplicației și detalii de implementare

3.1 Arhitectura aplicației

În acest subcapitol aș dori să evidențiez arhitectura aplicației „City Challenge”, precum și principalele module ale aplicației și modul în care acestea comunică.

În ceea ce privește arhitectura aplicației, aceasta este de tip client-server, după modelul pe două nivele în care primul strat este reprezentat de aplicația client și anume aplicația Android iar cel de-al doilea strat este serverul C++. Pentru stocarea datelor am utilizat biblioteca SQLite ce oferă o bază de date ușor de configurat și eficientă.

3.1.1 Baza de date

Baza de date este formată dintr-un număr de cinci tabele, structură redată în *Fig. 3.1*, unde tabela *users* este tabela în care se reține informațiile despre utilizatori înregistrați la aplicației iar tabela *maps* este cea în care sunt stocate datele despre hărțile disponibile, și anume o imagine reprezentativă pentru locația de start și numele acestei locații.

O altă tabelă este *markers* unde se păstrează toate informațiile legate de punctele de control de pe hărțile aplicației. În această tabelă se reține date precum *mapid*-ul hărții la care aparține punctul de control numit și *marker*, coordonatele x și y ale punctului pe harta oferită de API-ul Google Maps și întrebarea și răspunsul pentru ghicitoarea pe care utilizatorul trebuie să o rezolve în momentul în care ajunge la un punct de control.

Tabela *score* este tabela în care se reține cele mai bune rezultate cu privire la timpul în care se completează un anumit traseu.

Ultima tabelă, *usermaps*, este o tabelă de legătura între *maps* și *users* pentru a stoca harta disponibilă pentru fiecare utilizator și care este cel mai bun rezultat al utilizatorului pe harta respectivă.

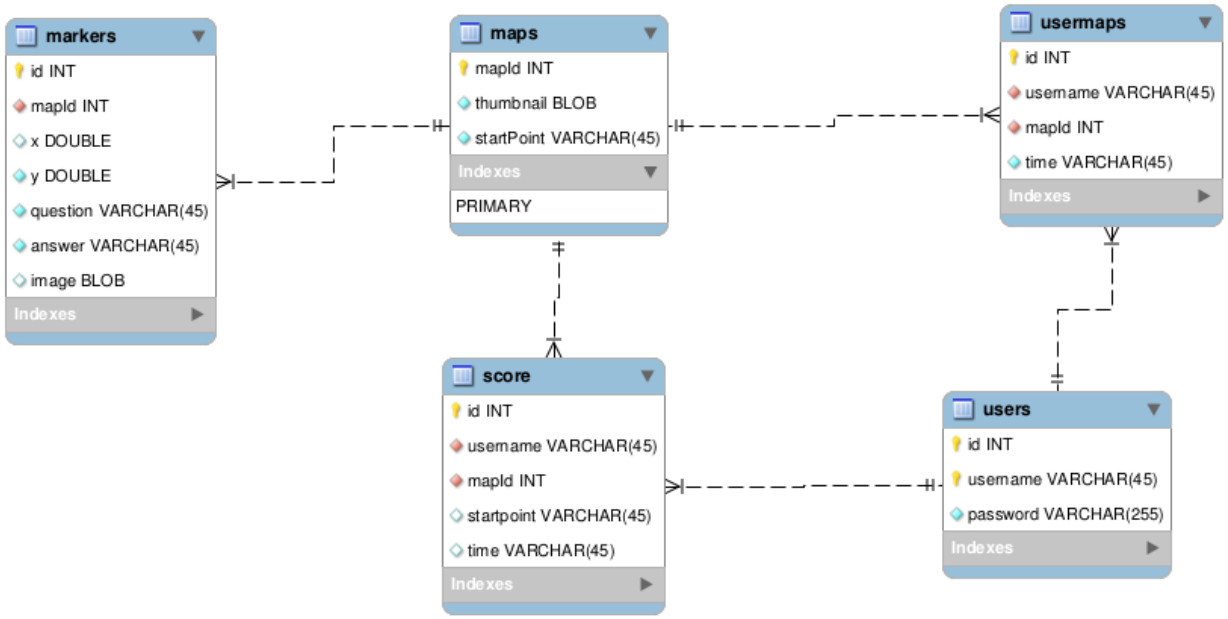

Fig. 3.1 Diagramă bază de date

3.1.2 Server

Pentru implementarea server-ului C++ am urmărit cât mai multe principii de programare orientată pe obiecte pentru a realiza o aplicație ușor de înțeles, modificat, întreținut și testat. Astfel, clasele încapsulează datele, ascund reprezentarea și sunt ușor de refolosit. De asemenea, responsabilitățile sunt alocate astfel încât coeziunea în sistem rămâne ridicată. O coeziune ridicată înseamnă că responsabilitățile pentru un element din sistem sunt înrudite și concentrate în jurul aceluiași concept. Totodată, sarcinile sunt împărțite în așa fel încât cuplarea rămâne slabă. O cuplare slabă presupune dependențe puține între clase, impact scăzut în sistem la schimbarea unei clase și potențial ridicat de refolosire.

În *Fig. 3.2* este reprezentată structura server-ului care este format din zece clase și anume: Clasa *ServerService,* care este clasa principală ce controlează întreaga logică a aplicației și realizează managementul tuturor celorlalte clase. Această clasă este realizată utilizând design pattern-ul *Singleton* care nu permite să existe în sistem mai mult de o singură instanță a acesteia. Clasa *ConnManager* se ocupă de tot ceea ce înseamnă comunicare cu clienții și conectare la aceștia cât și setări de configurare asupra socket-ului asociat server-ului. În ceea ce privește clasa *DbManager,* este cea care îndeplinește operațiunile de lucru cu baze de date: inserări, ștergeri, citiri și actualizări. Următoarea clasă, *Game*, este o clasă în care se rețin informații cu privire la un joc

în modul *multiplayer*, informații precum lista participanților la joc, harta pe care se joacă și numele jocului. Clasa *MessageCheck* este o clasă care se ocupă cu analizarea mesajelor primite de la client în vederea procesării acestora și mai apoi trimiterea informațiilor corecte înapoi către client. Clasele *UserManager* și *MapsManager* au ca rol managementul utilizatorilor pe server și respectiv a hărților. De asemenea, clasa *UserManager* este realizată conform design-pattern-ului *Singleton* astfel existând o singură instanța a acestei clase, pe când clasa *MapsManager* are câte o instanță la fiecare utilizator în parte pentru a reține informații cu privire la hărțile deținute de acel utilizator. O altă clasă este *User,* care conține datele despre utilizatori cât și managerul de hărți atașat fiecărui jucător. Clasa *Map* este o clasă ce reține pentru fiecare hartă punctul de plecare, imaginea sugestivă a acestuia și lista cu toate punctele de control de pe acel traseu. În cele din urmă, clasa *Marker* înglobează toate datele cu privire la un punct de control de pe un anumit traseu și anume, coordonatele pe hartă, întrebarea și răspunsul pentru ghicitoarea de la acest punct și o imagine dacă ghicitoarea conține o imagine.

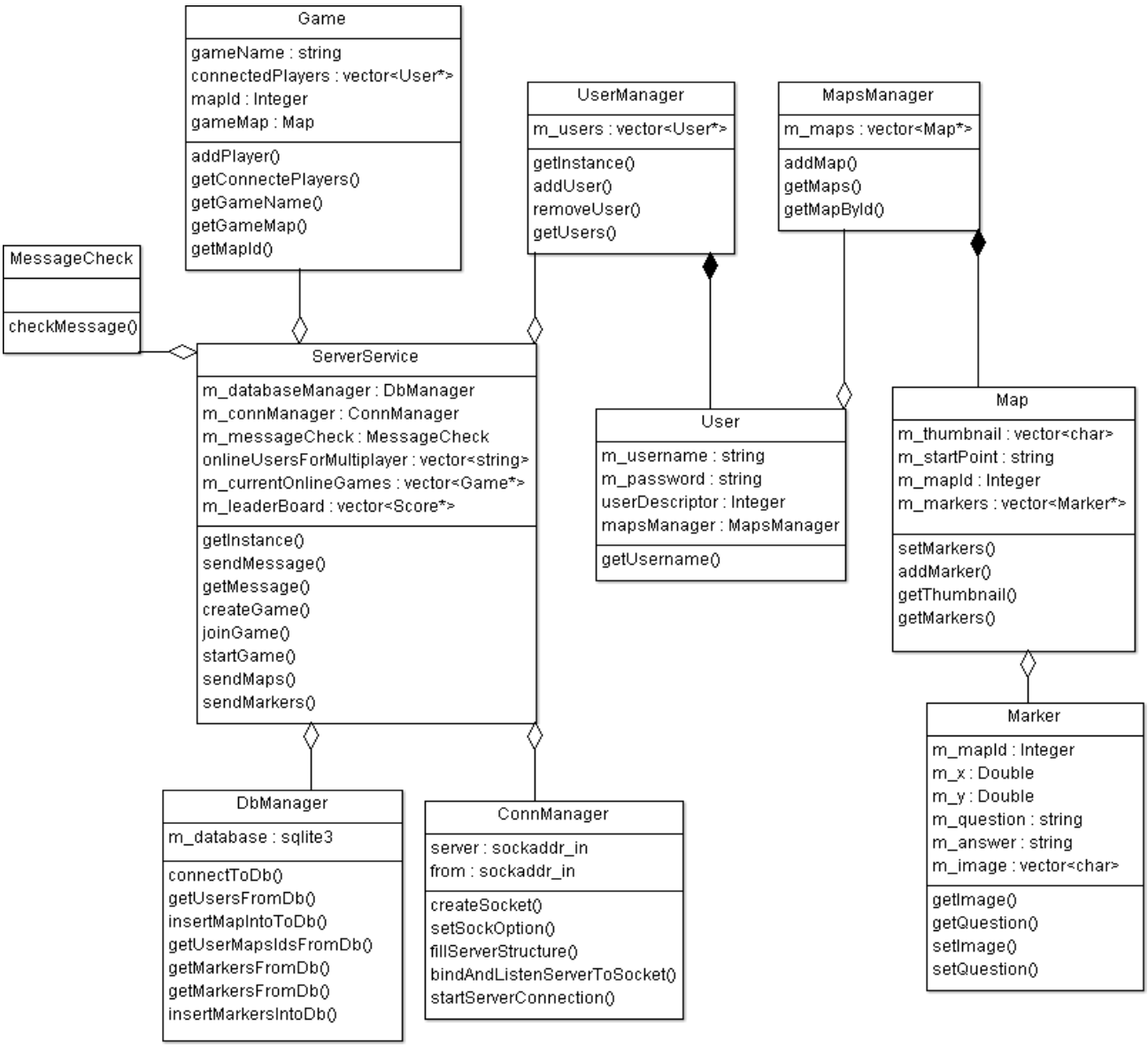

Fig. 3.2 Diagramă clase server

3.1.3 Client

În ceea ce privește arhitectura clientului, aceasta este alcătuită din patru mari module și anume: câte unul pentru fiecare mod de joc și unul care reține informați comune celorlalte trei module.

Primul modul, numit *SinglePlayer*, ilustrat în *Fig. 3.3*, se ocupă de crearea jocului în care există un singur participant și afișarea celor mai bune rezultate pe un anumit traseu. Acesta conține

patru clase după cum urmează: *singleplayer_StartActivity*, *selectmap_Activity*, *score_Activity*, *maps_Activity* și *puzzle_Activity*.

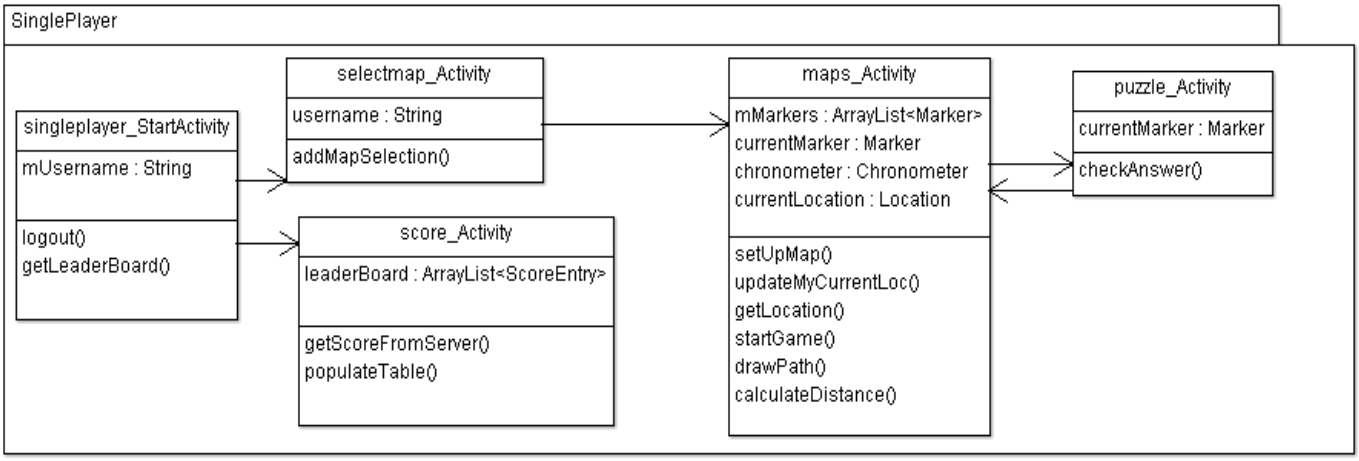

Fig. 3.3 Diagramă clase client modul SinglePlayer

Cel de-al doilea modul, numit *Multiplayer*, ilustrat in *Fig. 3.4,* se ocupă de crearea jocului în care pot exista mai mulți participanți și este alcătuită din șase clase: *multiplayerstart_Activity*, *onlineGamesSelect_Activity*, *selectMultiplayerMap_Activity*, *lobby_Activity*, *puzzle_Activity* și *multiplayerMaps_Activity*.

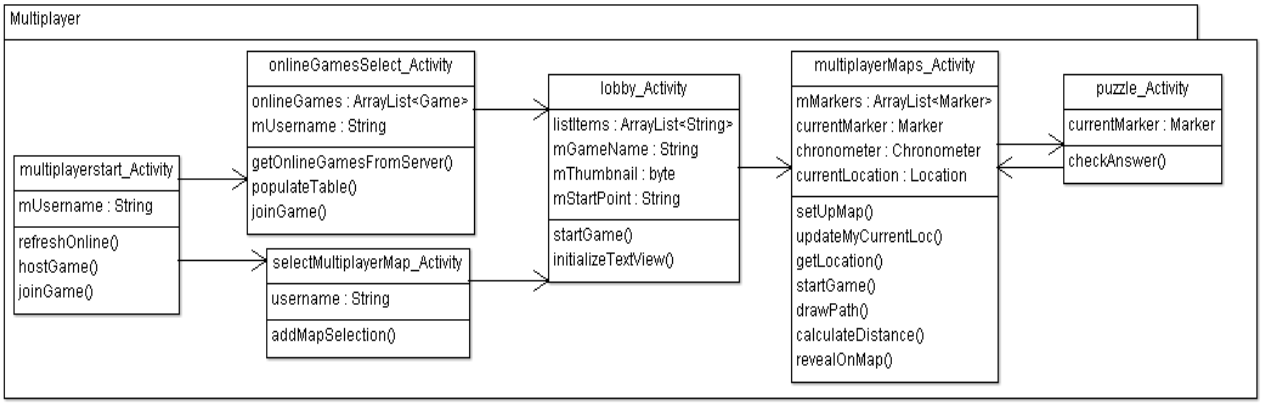

Fig. 3.4 Diagramă clase client modul Multiplayer

Modulul trei, numit *CreateMode*, ilustrat în *Fig. 3.5*, oferă utilizatorului posibilitatea să creeze un nou traseu și conține patru clase: *createmode_startActivity*, *startpoint_Activity*, *createMap_Activity*, *puzzleCreate_Activity*.

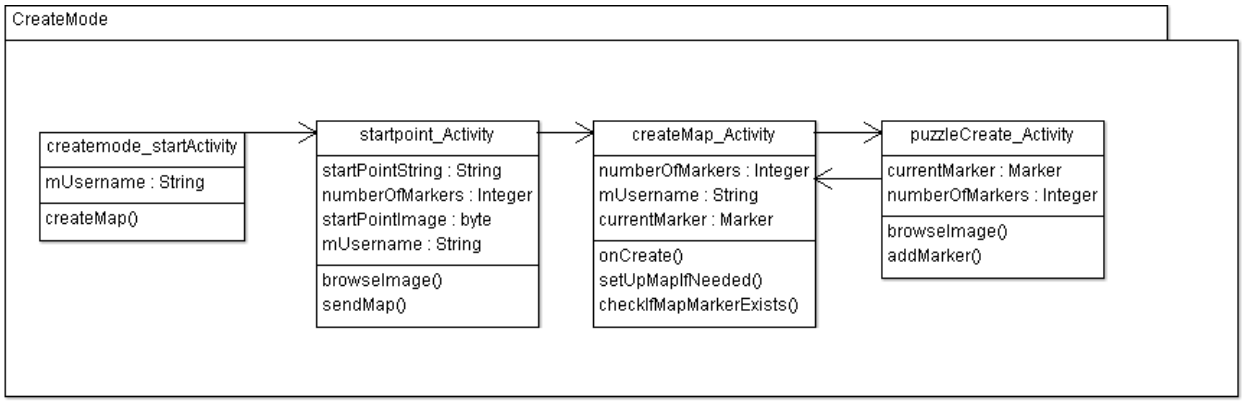

Fig. 3.5 Diagramă clase client modul CreateMode

Ultimul modul, este cel de început și realizează trecerea la unu din celelalte trei module prezentate mai sus conform *Fig. 3.6*. Acesta se numește *General* și este alcătuit din următoarele clase: *login_Activity, ConnManager, Map, MapMarker*.

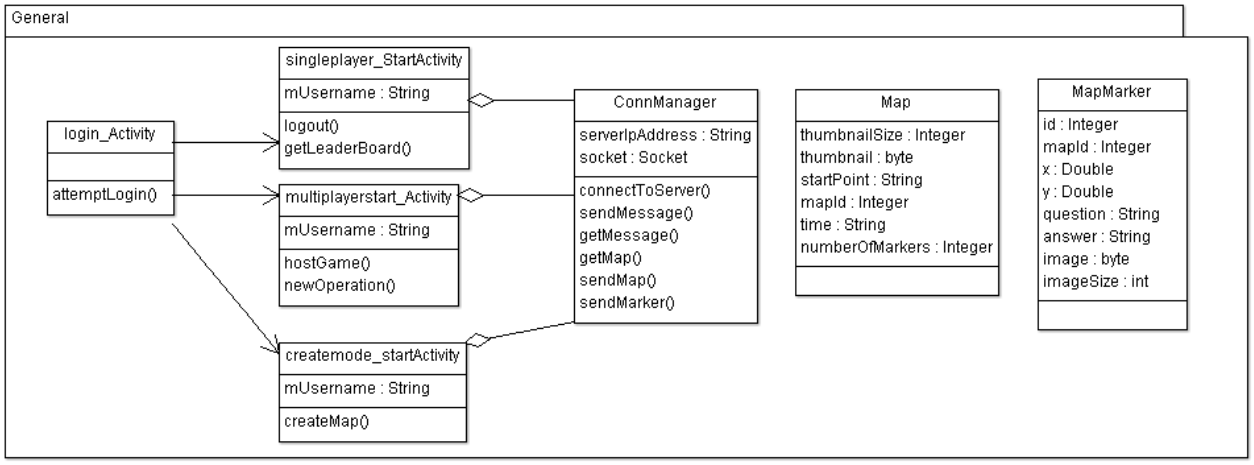

Fig. 3.6 Diagramă clase client modul General

3.2 Scenarii de utilizare

În acest subcapitol vom prezenta scenariile de utilizare posibile în aplicație, precum și imagini sugestive ale acestor scenarii.

3.2.1 Autentificare

Pentru a se conecta la server, înainte de a intra în aplicație, utilizatorul trebuie să activeze conexiunea la internet. Odată ce intră în aplicație, acesta va fi întâmpinat de un ecran de autentificare, iar în partea de jos a ecranului va apărea statusul server-ului și anume bulina roșie pentru închis, respectiv bulina verde pentru deschis conform *Fig. 3.7* și *Fig. 3.8.*

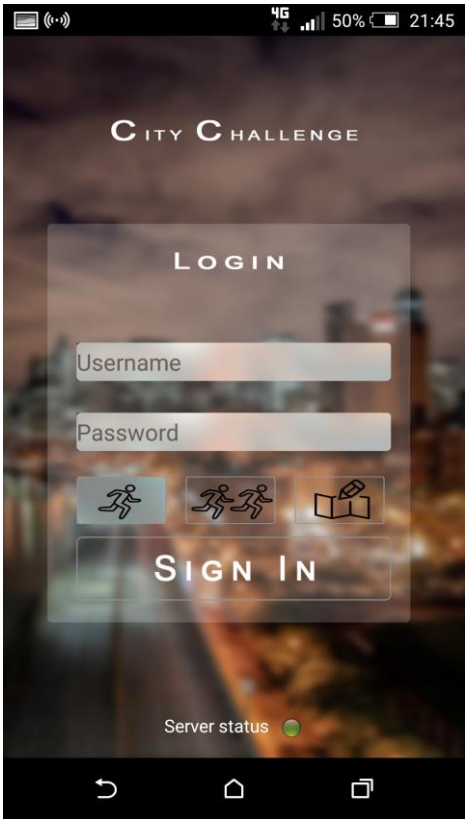

Fig. 3.7 Ecran autentificare, server deschis

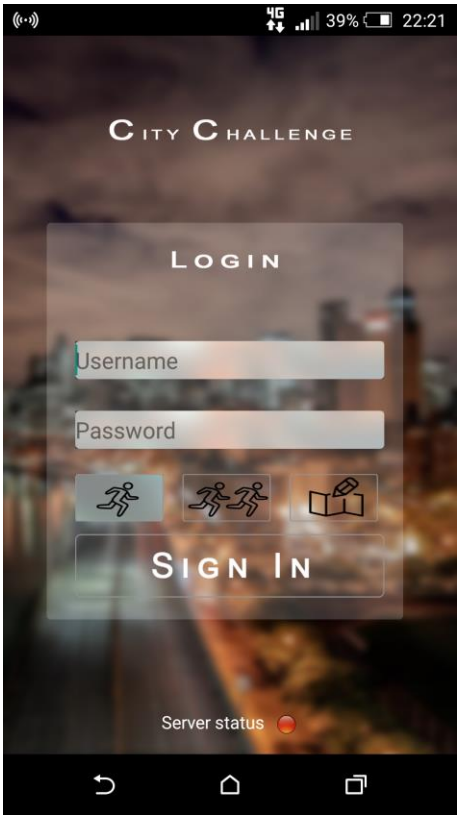

Fig. 3.8 Ecran autentificare, server închis

Dacă statusul server-ului este reprezentat printr-o bulina verde, atunci jucătorul poate introduce username-ul și parola apoi trebuie să selecteze un mod de joc după cum urmează: iconița din *Fig. 3.9* semnifică modul de joc *SinglePlayer*, iconița din *Fig. 3.10,* semnifică modul de joc *MultiPlayer,* iar iconița din *Fig. 3.11* reprezintă modul *CreateMode*. După selectarea modului de joc, dacă autentificarea s-a realizat cu succes, atunci utilizatorul va fi redirecționat către unul din cele trei moduri de joc, scenarii prezentate în capitolele ce urmează.

Fig. 3.9 SinglePlayer

Fig. 3.10 Multiplayer

Fig. 3.11 CreateMode

3.2.2 SinglePlayer

Pe pagina de *SinglePlayer*, utilizatorul va avea posibilitatea de a porni un joc, de a vizualiza cele mai bune rezultate pe diferite trasee în acest mod de joc sau să se întoarcă la ecranul de autentificare, aspect ilustrat in *Fig. 3.12.*

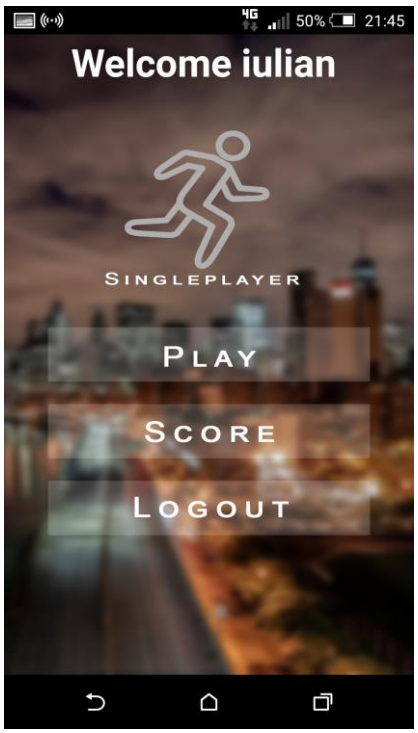

Fig. 3.12 Captură ecran modul singleplayer

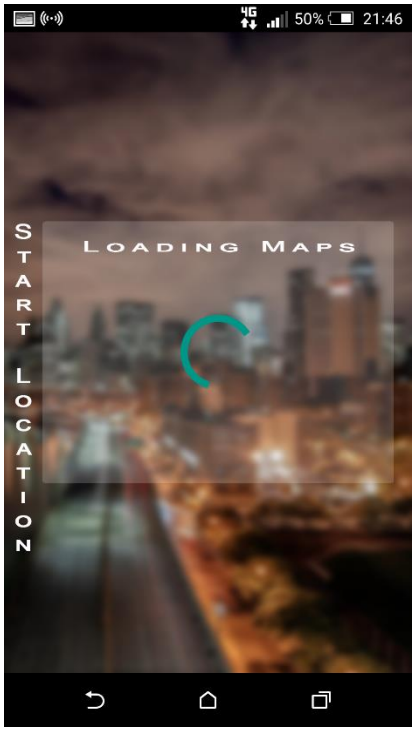

Fig. 3.13 Încărcare hărți

Dacă utilizatorul selectează opțiunea de a porni un nou joc atunci, după încărcare (*Fig. 3.13)*, se vor afișa acestuia traseele disponibile, mai exact o listă cu punctele de start de la fiecare traseu împreună cu cel mai bun timp scos pe acel traseu, numărul de puncte de control și denumirea punctului de start conform *Fig. 3.14.* Participantul la joc are posibilitatea de a apăsa pe o imagine aferentă unui traseu pentru a-l selecta. Odată selectat un anumit traseu, aplicația va încărca o hartă pe care se va pune ca și obiect de tip *Google Marker* punctul de start, aspect redat în *Fig. 3.15.*

Fig. 3.14 Selectare punct de start traseu

Fig. 3.15 Captură din timpul jocului singleplayer

Acesta poate vedea în partea de sus a ecranului cel mai bun rezultat al său pe traseul respectiv cât și timpul pentru jocul curent. În partea de jos, acesta are posibilitatea de reseta harta sau de a porni rezolvarea ghicitorii de la punctul de control curent. Dacă jucătorul nu se află destul de aproape de punctul de control curent, atunci nu va putea sa rezolve ghicitoarea și i se va oferi o sugestie de traseu către acest punct. Dacă, în schimb se află suficient de aproape, prin apăsarea butonului de „Resolve next puzzle”/„Start the first puzzle”, atunci aplicația va încărca puzzle-ul de la acel punct de control iar utilizatorul trebuie sa îl rezolve, aspect redat în *Fig. 3.16a/Fig 3.16b.* Rezolvarea corecta a ghicitorii va determina aplicația să încarce pe hartă următorul punct de control și de asemenea să sugereze un posibil traseu până acolo conform *Fig. 3.15*. Scenariul se repetă pentru fiecare astfel de punct. La final, utilizatorului îi va fi comunicat timpul realizat pe acest traseu iar daca este un timp mai bun decât cel de pana acum, aplicația va reține acest lucru.

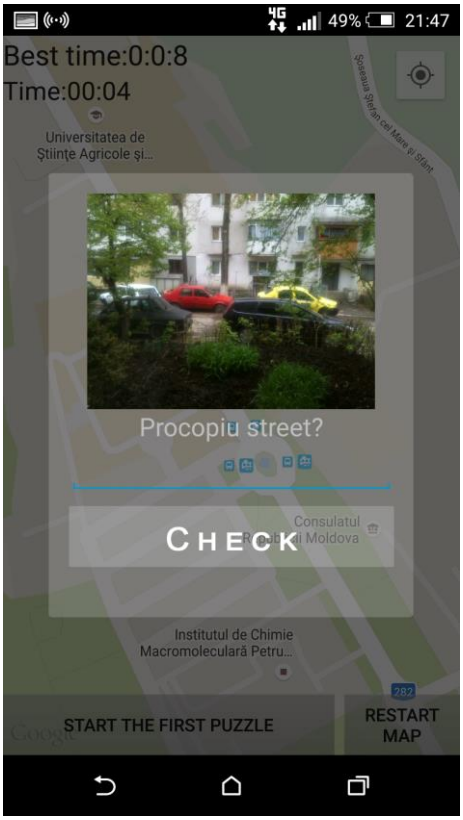

Fig. 3.16a Rezolvare puzzle

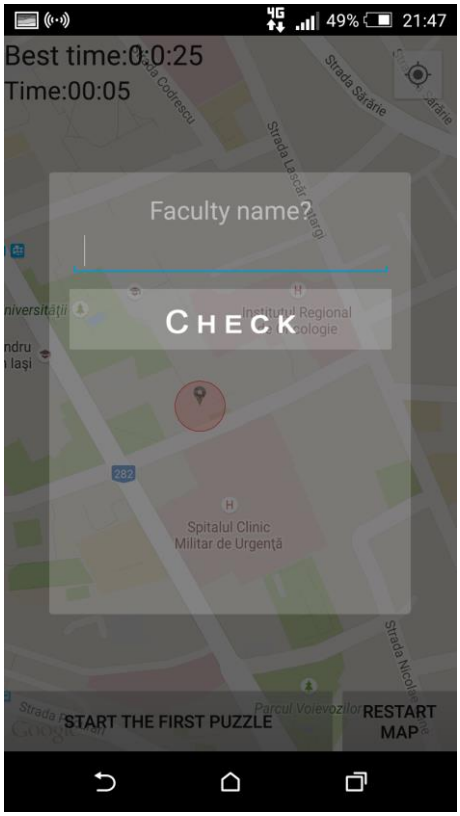

Fig. 3.16b Rezolvare puzzle

Prin selectarea opțiunii de a vizualiza cele mai bune rezultate, aplicația va încărca o listă cu cele mai bune astfel de timpuri împreună cu numele participantului la joc și numele traseului, conform *Fig. 3.17.*

Fig. 3.17 Tabela cu cele mai bune rezultate

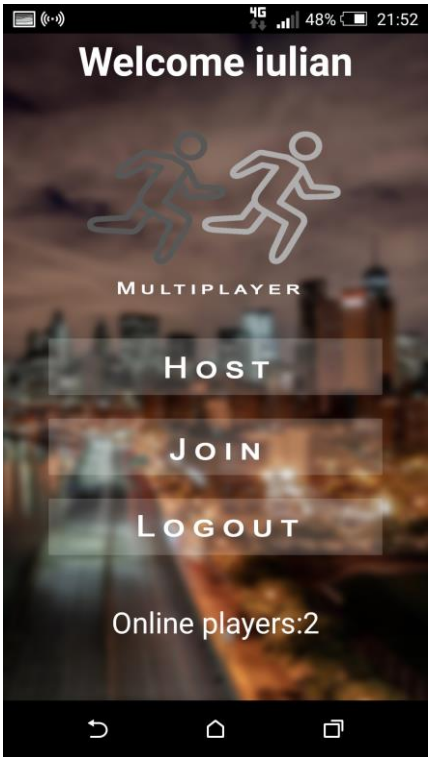

Fig. 3.18 Captură ecran modul multiplayer

3.2.3 Multiplayer

În ceea ce privește pagina de *MultiPlayer*, aceasta permite găzduirea unui joc nou pentru alți jucători, alăturarea la un joc deja găzduit de către altcineva sau întoarcerea la pagina de autentificare, aspect evidențiat in *Fig. 3.18.* Totodată, în partea de jos a ecranului apare numărul de utilizatori conectați în acest moment la acest mod de joc.

La selectarea opțiunii de găzduire a unui nou joc se vor încărca traseele disponibile similar modului *SinglePlayer* (*Fig. 3.14*). După selectarea unui anumit traseu aplicația creează pentru utilizator o cameră de joc pentru a aștepta alți participanți la joc să intre în această cameră după cum se poate vedea în *Fig. 3.19*, și nu poate începe jocul până când un alt jucător nu se alătură jocului. De asemenea, în acest moment jocul devine vizibil în meniul de ”join game”. Astfel, dacă alt utilizator selectează sa intre într-un joc deja creat aplicația îi va oferi o listă cu jocurile disponibile în acest moment, aspect redat în *Fig. 3.20.* Prin selectarea unuia din jocurile disponibile, acest utilizator va intra în camera de joc alături de cel care găzduiește jocul iar cel care a creat jocul poate selecta acum butonul ”Start” pentru a începe jocul acțiune evidențiată în *Fig. 3.21.*

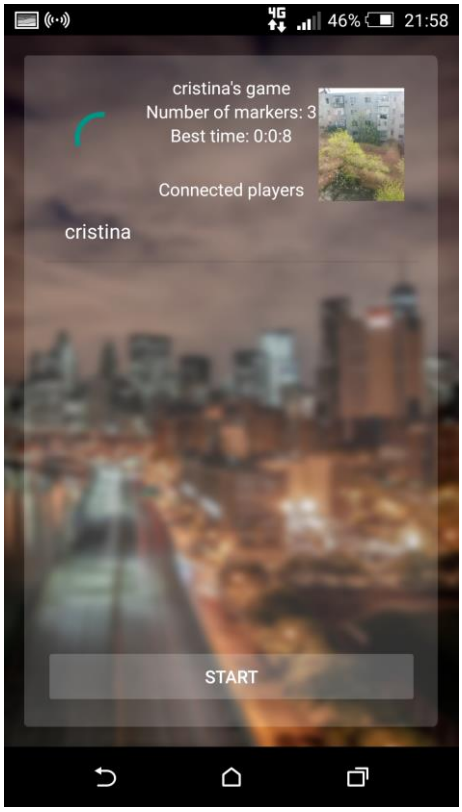

Fig. 3.19 Camera de joc

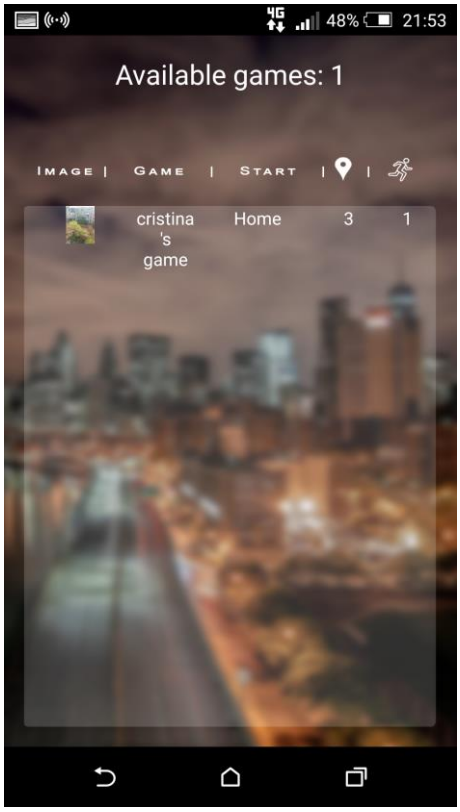

Fig. 3.20 Jocuri disponibile

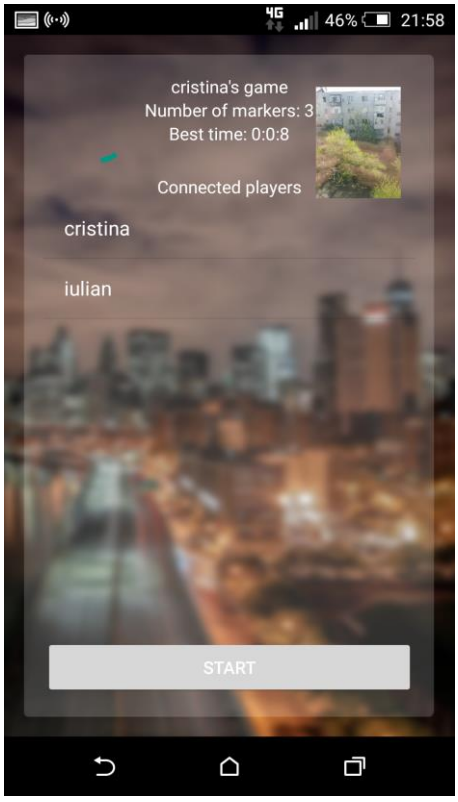

Fig. 3.21 Camera de joc/Start disponibil

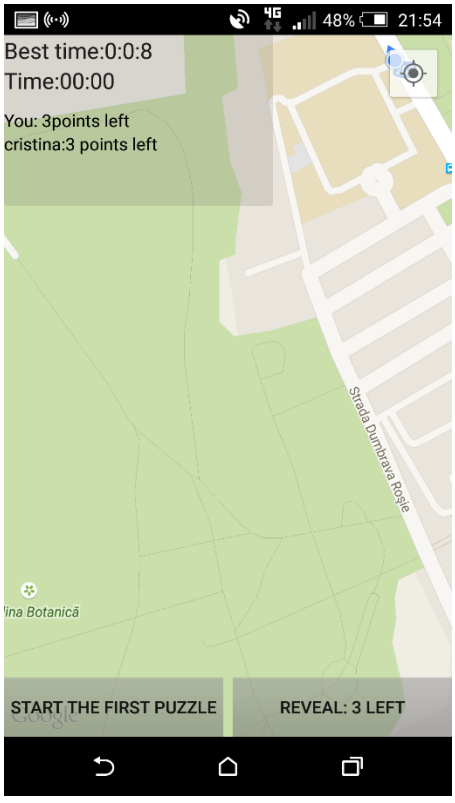

Fig. 3.22 Captură din timpul jocului multiplayer

Când cel care a găzduit jocul apasă butonul *Start*, aplicația va încărca harta ambilor participanți la joc în același timp. În timpul jocului, fiecare jucător poate vedea în ce stadiu se află față de adversar urmărind numărul de puncte de control pe care îl mai au fiecare pentru a termina, afișat în partea de sus a ecranului, precum și timpul curent, conform *Fig. 3.22*. De asemenea, un jucător are posibilitatea în timpul unui joc, de trei ori, să afișeze pe hartă locația rivalului său, prin apăsarea butonului *Reveal*. La apăsarea acestui buton, pe hartă va apărea un marker de culoare albastră reprezentând poziția celuilalt jucător, aspect evidențiat în *Fig. 3.23*. În momentul în care unul din participanți la joc termină jocul, adversarul său va fi notificat de acest lucru și de asemenea va fi anunțat faptul ca a pierdut jocul (*Fig. 3.24*). De asemenea, în mod similar, utilizatorii vor fi notificați dacă unul din partenerii de joc părăsește jocul.

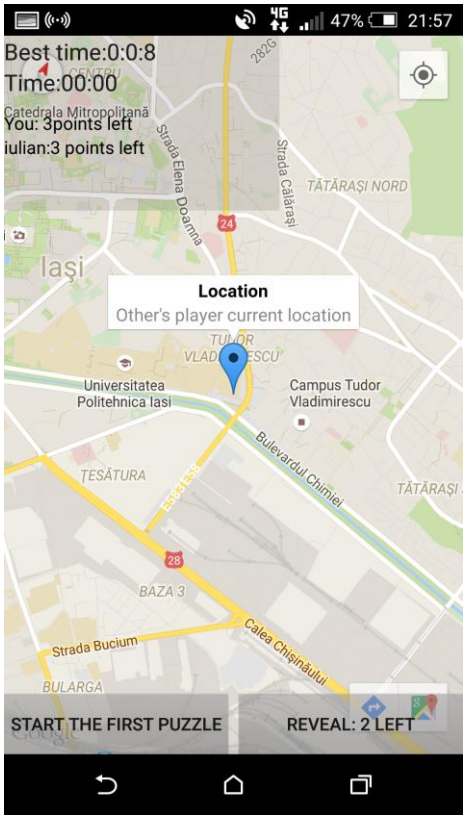

Fig. 3.23 Poziția adversarului

Fig. 3.24 Joc terminat de adversar

3.2.4 CreateMode

Pagina de *CreateMode* permite utilizatorului să creeze o noua hartă sau întoarcerea la pagina de autentificare, aspect evidențiat in *Fig. 3.25.*

Dacă utilizatorul alege să creeze un nou traseu, aplicația va încărca o pagină în care acesta trebuie sa selecteze punctul de start pentru acest traseu și anume: să selecteze o imagine reprezentativă pentru primul punct de control, numele acestuia și numărul de puncte de control pe care le va avea noul traseu, conform *Fig. 3.26.*

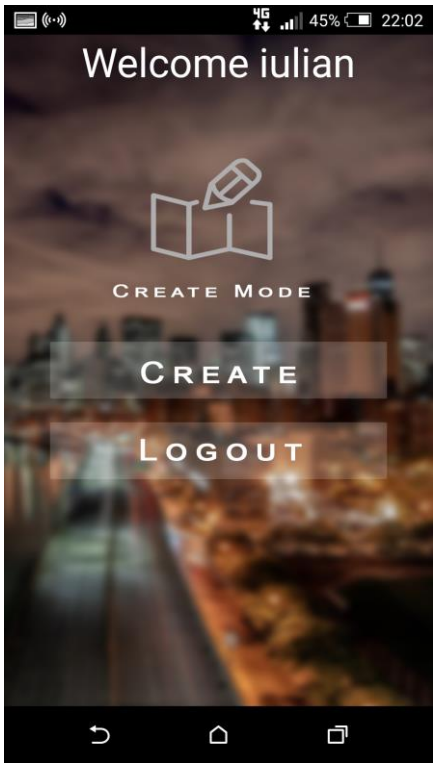

Fig. 3.25 Captură ecran modul createmode

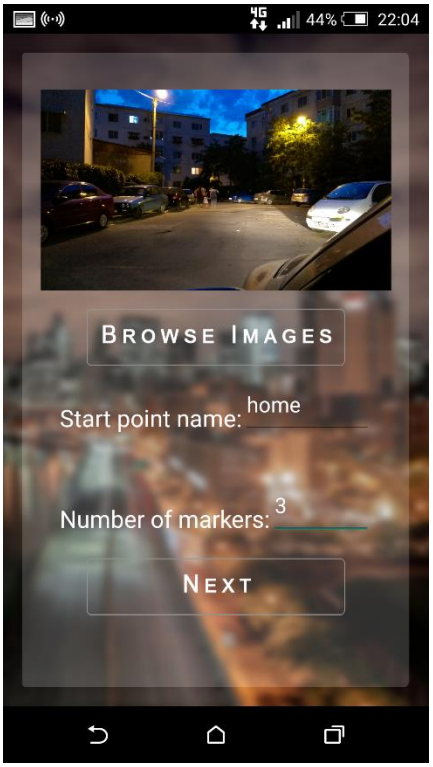

Fig. 3.26 Selectare punct de start traseu nou

Odată ce a configurat aceste cerințe, prin apăsarea butonului *Next* utilizatorului i se va încărca o hartă pe care trebuie să așeze punctele de control pe traseu. La apăsarea lungă pe hartă va apărea un punct care, folosind opțiunea de „Drag and Drop", poate fi mutat oriunde se dorește pe hartă, aspect evidențiat în *Fig. 3.27.* Totodată, prin apăsarea pe unul din aceste puncte, jucătorul poate să îl șteargă, să adauge la acest punct un puzzle sau să editeze puzzle-ul deja adăugat (*Fig. 3.28).*

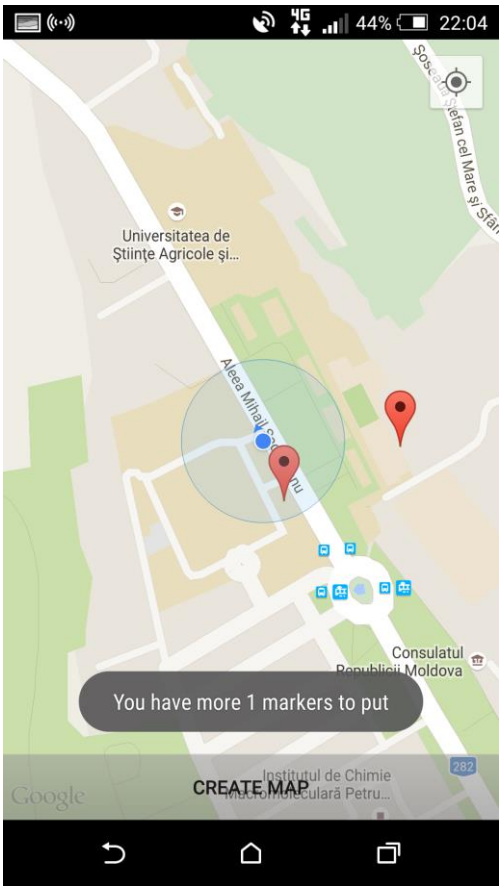

Fig. 3.27 Amplasare puncte de control

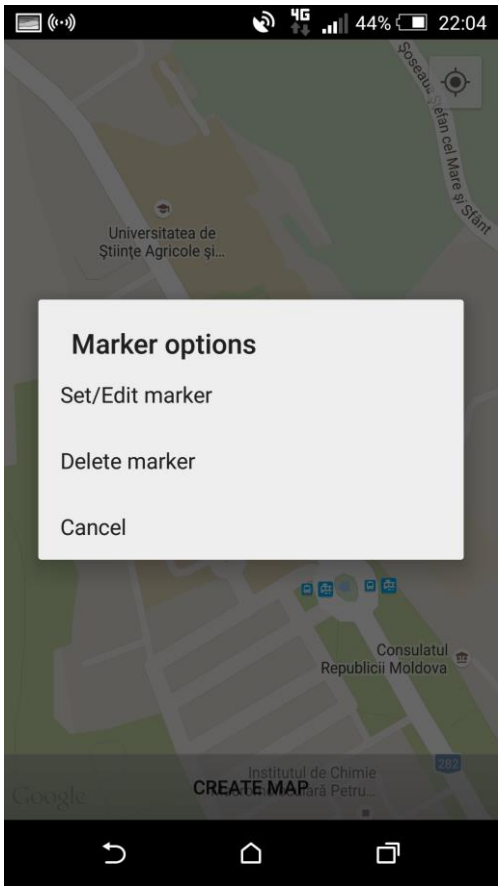

Fig. 3.28 Opțiuni pentru un punct de control

Dacă utilizatorul alege să seteze ghicitoarea pentru un anumit punct de control, aplicația va afișa o pagină în care acesta trebuie să selecteze întrebarea și răspunsul de la acest punct de control și opțional o imagine. De asemenea, acesta trebuie să selecteze care va fi ordinea de apariție a acestui punct de control pe hartă în timpul jocului, aspect evidențiat în *Fig. 3.29*.

După ce toate punctele de control au fost setate corespunzător, noul traseu va fi creat iar utilizatorul este invitat în modul *SinglePlayer* să testeze acest nou traseu, conform *Fig. 3.30*.

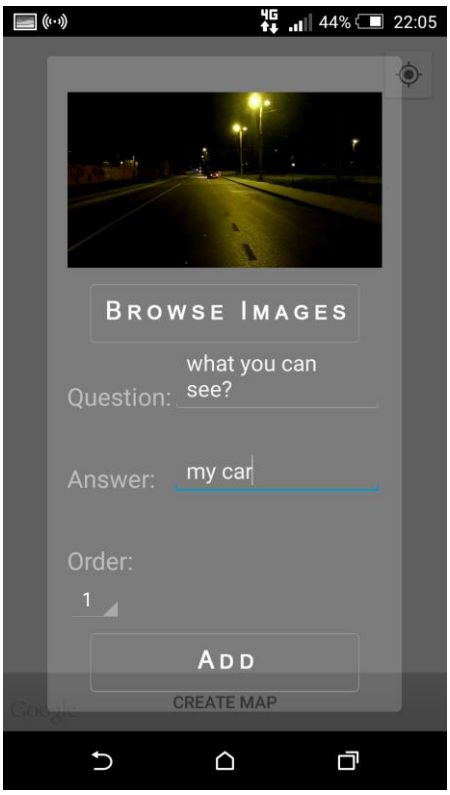

Fig. 3.29 Configurare puzzle punct de control

Fig. 3.30 Noul traseu a fost creat

3.3 Detalii de implementare

3.3.1 Server

La nivel de implementare, serverul va fi un proces pornit permanent. Înainte de a inițializa conexiunea cu clienții, serverul realizează conexiunea la baza de date, încarcă toate datele din aceasta și creează obiecte aferente fiecărei informații preluate, conform arhitecturii. Astfel, pentru utilizatori se vor crea obiecte de tip *User*, iar fiecare va avea atașat cate un obiect de tip *MapsManager* care conține instanțe de tip *Map* ce vor reprezenta hărțile încărcate din baza de date. La rândul lor, acestea vor avea încapsulate obiecte de tip *Marker* ce reprezintă punctele de control de pe fiecare traseu. Instanțele de tip *User* vor fi stocate într-un vector din librăria STL a C++ care va fi reținut în clasa *UserManager* a cărei instanță va fi unică utilizându-se design pattern-ul *Singleton*.

Un avantaj al încărcării tuturor datelor la pornirea serverului este acela de a reduce cât mai mult accesul fizic la baza de date deoarece poate fi foarte costisitor în anumite cazuri.

După încărcarea datelor, serverul, folosind clasa *ConnManager*, creează socket-ul de conectare, îi atribuie o adresă și setează pe socket câteva opțiuni specifice precum SO_REUSEADDR pentru a refolosi același port în cazul unui restart și SO_SNDBUF prin care se modifică dimensiunea buffer-ului alocat socket-ului la o dimensiune mai mare, datorită transferului de imagini care se va realiza pe acest socket.

După ce aceste operațiuni au fost realizate cu succes, serverul întră într-o stare de așteptare până la conectarea unui client folosind primitiva *accept()*. Când se realizează acest lucru, serverul creează un fir de execuție nou ce se va ocupa de comunicarea cu acesta reușind astfel să trateze simultan oricât de mulți clienți conectați conform *Listing nr. 1*.

Listing nr. 1

```
            if ( (*clientDescriptor = accept (serverDescriptor, (struct
sockaddr *) &from, &length)) < 0)
            {
                  cout<<"[Server]Error on accept().\n";
                  continue;
            }
            cout<<"Descriptor is "<<*clientDescriptor;
            cout<<"Creating thread for client"<<endl;
            i++;

      pthread_create(&th[i],NULL,&ServerService::treat,clientDescriptor);
```

Astfel, începe comunicarea efectivă între client și server ce presupune citirea mesajului primit de la client, interpretarea acestuia într-o operațiune validă pe care server-ul o poate realiza și oferirea răspunsului corespunzător. Fiecare mesaj de la client este citit utilizând paradigma *read()* de la descriptorul de socket asociat clientului respectiv, aspect evidențiat in *Listing nr. 2*, iar mesajele sunt trimise prin metoda *write()* de asemenea utilizând descriptorul de socket asociat, conform *Listing nr.3* . Orice mesaj primit de la client este convertit, utilizând paradigma *ntohl()*, la tipul de arhitectură al procesorului pe care rulează serverul, în cazul de fața Intel, deci Little-endian.

Listing nr. 2

```
bool ServerService::getMessage(int descriptor,string *message){
      int messageSize;
      int returnCode;
      if((returnCode = read(descriptor,&messageSize,sizeof(messageSize))) ==
0){
            logMessage("Clientul s-a deconectat");
            return false;
      }
      messageSize = ntohl(messageSize);
      char messageRead[messageSize];
      if((returnCode = read(descriptor,messageRead,messageSize)) == 0){
            logMessage("Clientul s-a deconectat");
            return false;
      }
      else{
            messageRead[returnCode] = '\0';
            logMessage("Bytes read: "+to_string(messageSize));
            //logMessage(messageRead);
            (*message).assign(messageRead);
            return true;
      }
}
```

Listing nr. 3

```
bool ServerService::sendMessage(int descriptor, string message){
      logMessage(message);
      int* messageSize = new int(message.length());
      write(descriptor,messageSize,sizeof(messageSize));
      char messageToSend[types::BUFSIZE];
      strcpy(messageToSend,message.c_str());
      write(descriptor,messageToSend,strlen(messageToSend));
      return true;
}
```

Pentru a analiza fiecare mesaj primit de la client, am creat clasa *MessageCheck* care examinează acest mesaj și returnează un tip *enum* aferent tipului de operațiune suportat de server. Astfel, în funcție de mesajul returnat de această clasă, serverul realizează acțiunea necesară și returnează rezultatul către client.

Pentru trimiterea oricărui mesaj către client, inițial se trimite numărul de biți care urmează a fi scriși și apoi mesajul efectiv întrucât clientul sa primească mesajul corect.

În subcapitolele următoare vom prezenta toate acțiunile realizate de server în fiecare scenariu de utilizare.

Autentificare/Înregistrare

În cazul în care utilizatorul solicită autentificarea în aplicație, serverul verifică în vectorul din clasa *UserManager* dacă există un utilizator cu username-ul și parola primită iar în caz afirmativ îi setează acestuia descriptorul de socket aferent și trimite către client mesajul de succes. În caz negativ, serverul va trimite către client un mesaj de autentificare eșuată. În mod similar, dacă utilizatorul alege să se înregistreze în aplicație serverul va primi username-ul și parola de la client, care va trimite aceste date către *DbManager* ce va realiza inserarea în baza de date a noului utilizator urmând ca apoi să se trimită către client mesajul aferent de succes sau nu.

SinglePlayer

Dacă utilizatorul alege modul de joc SinglePlayer, clientul poate cere de la server lista cu hărțile disponibile sau tabela cu cele mai bune rezultate.

La cererea hărților disponibile prin primirea mesajului *„getmaps:username”*, serverul va trimite, rând pe rând, toate hărțile pe care un utilizator le are asociate. Mesajele ce conțin hărțile vor începe prin mesajul „*maps:start*” urmate apoi de numărul de hărți și apoi pentru fiecare astfel de hartă se va trimite locația de start, id-ul hărții, timpul realizat pe acest traseu, numărul de puncte de control și imaginea sugestivă. Imaginile asociate fiecărui traseu sunt salvate în baza de date sub formă de BLOB (Binary Large OBject). Este apoi încărcat în memorie și reținut într-un tip *array* de *unsigned char*. Pentru a trimite această imagine către client, se trimite inițial dimensiunea acesteia urmată de un pointer către *array*-ul ce reprezintă imaginea, aspect evidențiat în *Listing nr. 4*.

Listing nr. 4

```
written =
write(clientDescriptor,sizeThumbnail,sizeof(sizeThumbnail));//dimensiune
imagine
logMessage("written size "+to_string(written));
written = write(clientDescriptor,byteArray,sizeof(byteArray));//pointer
către imagine
logMessage(written);
```

După ce clientului i se vor trimise toate traseele și își alege unul din acestea, serverul va primi mesajul de a trimite toate punctele de control pentru harta respectivă. Similar hărților, se va trimite inițial mesajul „*markers:start*” întrucât clientul să știe că urmează să primească punctele de control. Apoi, serverul trimite numărul de puncte ce urmează a fi trimise și apoi rând pe rând toate informațiile legate de fiecare punct de control și anume: coordonatele pe hartă, întrebarea și răspunsul ce formează ghicitoarea de la acest punct și opțional se poate trimite și o imagine dacă aceasta există. Dacă imaginea nu există, mesajul trimis va fi „*markers:finish*” pentru a marca terminarea trimiterii punctelor de control.

În momentul în care un jucător termină un traseu, serverul va primi mesajul de a actualiza timpul pe acel traseu. Serverul trimite mai departe comanda către *DbManager* care în funcție de timpul curent pe care îl are utilizatorul poate realiza actualizarea în baza de date dacă este un rezultat mai bun. De asemenea, dacă este un timp mai bun decât cel precedent, acesta este salvat în tabela cu cele mai bune rezultate.

Dacă la intrarea în modul de joc *SinglePlayer*, utilizatorul selectează să vizualizeze cele mai bune rezultate atunci serverul va primi mesajul de a trimite toate aceste rezultate existente. Similar hărților și punctelor de control, inițial se trimite mesajul „*score:start*” urmând apoi numărul de rezultate și informațiile despre fiecare rezultat în parte precum: numele traseului, timpul realizat și numele participantului care a realizat acest timp.

MultiPlayer

La alegerea utilizatorului de a intra în modul de joc *MultiPlayer*, serverul va primi mesajul *„multiplayerstart:username”*. În acest moment, în vectorul reținut în clasa *ServerService* cu toți

utilizatori care sunt intrați în modul de joc *Multiplayer* se mai adaugă și acest utilizator. De asemenea, serverul va recepționa un mesaj de tipul „*updateonline:username*”, în urma căruia se va trimite către client dimensiunea vectorului cu utilizatori conectați la acest mod de joc, întrucât clientul să fie notificat de numărul de jucători conectați în acel moment.

Dacă, în continuare, jucătorul alege să găzduiască un nou joc, serverul va primi mesajul „*getmaps:username*” în care clientul cere toate hărțile disponibile. Similar modului *SinglePlayer,* se vor trimite rând pe rând toate informațiile despre fiecare hartă în parte. După ce utilizatorul își alege una din hărți pentru joc, serverul va primi mesajul *„creategame:username:mapid”*. Din acest mesaj se extrage id-ul hărții dorite și pe baza acestuia, serverul creează un obiect de tip *Game* care va fi adăugat în lista de jocuri disponibile în acest moment pe server, aspect evidențiat în *Listing nr. 5.* Acest obiect deține o listă de jucători conectați la acest joc, în care va fi adăugat creatorul jocului. Odată ce utilizatorul intră în camera de joc, serverul va primi cererea de a trimite jucătorii conectați la jocul tocmai creat iar serverul va trimite numele acestor jucători, în cazul de față existând un singur jucător conectat și anume creatorul său.

Listing nr. 5

```
void ServerService::createGame(string username, int mapId, User*
currentUser){
	string gameName = username+"'s game";
	Game* newGame = new Game(gameName, mapId, currentUser);
	m_currentOnlineGames.push_back(newGame);
}
```

În cazul în care utilizatorul alege să se alăture unui joc deja creat, serverul va primi mesajul *„getonlinegames”* urmând ca apoi să trimită detaliile despre fiecare joc în parte și anume: imaginea reprezentativă pentru acel traseu, numele punctului de start, cel mai bun timp al creatorului jocului, numărul de puncte de control și numărul de jucători conectați la acel joc. Trimiterea se realizează similar trimiterii hărților, prin apeluri ale primitivei *write().* După ce participantul la joc alege un joc din lista trimisă, serverul va recepționa mesajul *„joingame:username:gamename”*. Din acest mesaj se extrage numele jucătorului și la ce joc dorește să se alăture. Astfel, se adaugă în lista de utilizatori conectați a jocului cu numele primit, utilizatorul acesta, și se trimite mesaj către cel care a creat jocul să își actualizeze lista de jucători conectați la jocul său. Dacă jocul deja a fost pornit sau există deja un număr de doi jucători conectați la acest joc se va trimite către client mesajul

„rejected”. În caz contra, este trimis mesajul *„succes”,* iar clientul poate intra în camera de joc. Astfel, după ce utilizatorul intră în camera jocului selectat, serverul va primi cererea de a trimite jucătorii conectați la acest joc.

Odată ce creatorul jocului apasă butonul de start, serverul va recepționa mesajul *„startgame:gamename”*. În acest moment, se va trimite același mesaj către jucătorul care s-a alăturat la acest joc pentru a porni și el jocul. De asemenea, pentru a reține câte puncte de control mai are de completat pe hartă fiecare jucător, în obiectul de tip *Game* se reține un obiect de tip *map* unde cheia este o instanță a clasei *User,* iar valoarea este numărul de puncte de control rămase. Astfel, la începerea jocului se completează acest obiect cu perechi de tipul: utilizator, număr total puncte de control existente pe hartă.

În momentul în care se încarcă harta la ambii jucători conectați la un joc, serverul va primi de la aceștia mesajul „*getothermarkers:username:gamename”* urmând ca apoi să se caute în lista jocului curent numărul de puncte de control ale adversarului și să se trimită. Totodată, pe parcursul jocului, când unul din cei doi participanți la joc completează un punct de control și trece la următorul, serverul va fi notificat de acest lucru pentru actualizarea numărului punctelor de control ramase pentru acest jucător în obiectul de tip *map* al jocului curent. De asemenea, serverul va notifica și adversarul său, prin trimiterea unui mesaj ce conține numărul de puncte de control ale acestuia. Dacă în acest timp unul din jucătorii conectați părăsește jocul, serverul va primi mesajul „*deletegame:gamename”* și astfel jocul va fi șters din lista jocurilor curente deoarece nu mai este un joc valid. De altfel, se va notifica și partenerul de joc că acesta a părăsit jocul.

În timpul jocului, participanții au posibilitatea de a afla poziția curentă pe hartă a adversarilor săi. La alegerea de către un jucător a acestei opțiuni, serverul va primi comanda „*getotherlocation:gamename:username”*. În acest moment, serverul trimite cererea de oferire a locației curente către adversarul său. Când acesta răspunde către server cu locația sa, serverul înaintează aceasta locație către primul jucător care a selectat opțiunea de vizualizare a locației partenerului de joc.

CreateMode

Dacă utilizatorul alege să creeze un traseu nou, după ce selectează detaliile legate de noul traseu, serverul va primi mesajul „*receivemap”*. În acest moment, serverul va citi de la client informațiile selectate de acesta. Astfel, pentru a citi imaginea reprezentativă a traseului, se va citi inițial dimensiunea acesteia, apoi, folosind aceasta dimensiune, utilizăm o bucla de tip *while* pentru

a fi siguri că se citesc toți biți care aparțin imaginii, aceasta buclă citind până când s-a recepționat întreaga imagine, aspect evidențiat în *Listing nr. 6.*

Listing nr. 6

```
       thumbnailSizeInt = ntohl(thumbnailSizeInt);
       logMessage("Read char: "+to_string(thumbnailSizeInt));
       unsigned char thumbnail[thumbnailSizeInt];
       int bytesLeftToRead = 0;
       while(bytesLeftToRead < thumbnailSizeInt){
             if((returnCode =
read(clientDescriptor,thumbnail+bytesLeftToRead,thumbnailSizeInt -
bytesLeftToRead)) == 0){
                   logMessage("Clientul s-a deconectat");
                   return false;
             }
             else{
                   bytesLeftToRead += returnCode;
                   logMessage("Bytes left: "+to_string(thumbnailSizeInt -
bytesLeftToRead));
             }
       }
```

După ce imaginea s-a primit, se citește și numele punctului de start și apoi se trimit toate informațiile acestea către *DbManager* care inserează noul traseu creat în baza de date. Pentru a evita atacuri de tip *SQLInjection,* la inserarea în baza de date a noii hărți se folosește conceptul de „prepared statements”, astfel că toate instrucțiunile de inserare sunt parametrizate, conform *Listing nr. 7.*

Listing nr. 7

```
returnCode = sqlite3_prepare_v2(m_database,"INSERT INTO
maps(thumbnail,startPoint)"
" VALUES(?,?)", -1, &stmt, NULL);
       if (returnCode != SQLITE_OK) {
             cout<< "prepare failed: " << sqlite3_errmsg(m_database) << endl;
       } else {
       returnCode = sqlite3_bind_blob(stmt, 1, thumbnail, thumbnailSize,
SQLITE_STATIC);
```

```
      returnCode =
sqlite3_bind_text(stmt,2,startPoint.c_str(),startPoint.size(),0);
            if (returnCode != SQLITE_OK) {
                  cout << "bind failed: " << sqlite3_errmsg(m_database) <<
endl;
            } else {
                  returnCode = sqlite3_step(stmt);
                  if (returnCode != SQLITE_DONE)
                     cout << "execution failed: " <<
sqlite3_errmsg(m_database) << endl;
            }
      }
      sqlite3_finalize(stmt);
```

După ce inserarea s-a realizat cu succes, id-ul hărții noi create se trimite către client. Odată ce toate punctele de control au fost setate pe client, detaliile despre fiecare astfel de punct se primesc de către server împreună cu id-ul traseului la care aparțin. Similar citirii datelor despre o noua hartă, se citesc și datele despre noile puncte de control asociate hărții, urmând ca apoi sa fie inserate în baza de date.

În cele din urmă, după ce atât harta cât și punctele de control aparținând acesteia au fost inserate în baza de date, ele sunt actualizate și la nivel de server, fără a fi nevoie de un restart să le încarce din baza de date. Astfel dacă un jucător dorește să încerce noua hartă creată, aceasta este disponibilă.

3.3.2 Client

În ceea ce privește implementarea clientului, acesta este o aplicație dezvoltată pe platforma Android.

Atât pentru conexiunea la server cât și pentru primirea și trimiterea de informații către acesta am creat clasa *ConnManager* utilizând design pattern-ul *Singleton* pentru ca aceasta conexiune la server să fie unică și să poate fi accesată din orice altă activitate. În această clasa, conexiunea la server se realizează prin crearea unui socket și conectarea acestuia la adresa și portul serverului, aspect redat în *Listing nr. 8*.

Listing nr. 8

```
private ConnManager(){
        try{
            serverIpAddress = Types.IP;
            serverAddr = InetAddress.getByName(serverIpAddress);
            addr = new InetSocketAddress(serverAddr, Types.PORT);
            socket = new Socket();
        } catch (Exception e) {
            Log.d("ConnManager()",e.toString());
        }
    }
  public boolean connectToServer(){
        try {
            Log.d("Connecting to socket","c");
            socket.setReceiveBufferSize(65536);
            socket.connect(addr, 3000);
            return true;
        } catch (Exception e) {
            Log.d("connectToServer",e.toString());
            if(e.getMessage().equals("Already connected")){
                return true;
            }
            return false;
        }
    }
```

Citirea și scrierea se realizează folosind *InputStream* și respectiv *OutputStream* puse la dispoziție de socket-ul creat. La scriere, similar serverului, se trimite inițial numărul de biți care urmează a fi trimiși apoi biți efectiv, aspect evidențiat în *Listing nr. 9.*

Listing nr. 9

```
public void sendMessage(String message){
        try {
            Log.d("In send message", message);
            OutputStream out = socket.getOutputStream();
```

```
            ByteBuffer b = ByteBuffer.allocate(4);
            b.putInt(message.length());
            byte[] sizeMessage = b.array();
            out.write(sizeMessage);
            out.write(message.getBytes());
        } catch (Exception e) {
            Log.d("sendMessage",e.toString());
        }
    }
```

Pentru citire, se preia un număr de opt biți și se ordonează conform arhitecturii Little Endian, apoi se extrage o variabilă de tip *int* din acești biți. Acest număr reprezintă numărul de biți care îl are mesajul. Utilizând o buclă de tip *while* se vor citi biți corespunzători mesajului apoi se va face conversia acestuia în tipul *string*, conform *Listing nr. 10*. De asemenea, dacă în citirea mesajului apare o eroare aceasta este tratată utilizând un bloc de tip *try catch* pentru a surprinde aceasta excepție și a returna un mesaj corespunzător.

Listing nr. 10

```
public String getMessage(){
        String reply = "";
        try {
            InputStream in = socket.getInputStream();
            byte[] sizeOfMessage = new byte[8];
            int count = in.read(sizeOfMessage,0,8);
            ByteBuffer buffer2 = ByteBuffer.wrap(sizeOfMessage);
            buffer2.order(ByteOrder.LITTLE_ENDIAN);
            int sizeOfMessageInt = buffer2.getInt();
            if(sizeOfMessageInt < 0 ){
                return reply;
            }
            byte[] message = new byte[sizeOfMessageInt];
            byte[] auxBufferforMessage = new byte[sizeOfMessageInt];
            int bytesLeftToRead = 0;
            while(bytesLeftToRead < sizeOfMessageInt){
                count = in.read(auxBufferforMessage,0,(sizeOfMessageInt-
bytesLeftToRead));
```

```
System.arraycopy(auxBufferforMessage,0,message,bytesLeftToRead,count);
                bytesLeftToRead += count;
            }
            String messageString = new String(message);
            reply = messageString;
            message = null;
            auxBufferforMessage = null;
        }
        catch (Exception e) {
            return "error";
        }
        Log.d("Receieved message:",reply);
        return reply;
    }
```

În mod similar, în clasa *ConnManager*, sunt implementate funcțiile care primesc sau trimit detaliile legate de trasee sau puncte de control.

Pentru o mai bună organizare a aplicației am creat trei pachete ce includ activitățile specifice fiecărui mod de joc. Aspectul grafic al fiecărei activități este dat de un fișier de tip *XML* propriu din directorul *res/layout* al aplicației. Imaginile și iconițele utilizate în aplicație de asemenea se află în directorul *re/drawable* al aplicației și acestea sunt fie creație personală fie preluate sub licență de tip *Free for non-commercial use*[18].

În subcapitolele următoare vom prezenta toate acțiunile realizate de client în fiecare scenariu de utilizare.

Autentificare/Înregistrare

La intrarea în aplicație a utilizatorului se va încărca, din pachetul *General*, activitatea login*_Activity* care conține câmpurile de introducere a username-ului și a parolei sub forma de *EditText* și trei butoane de tip *radio* ce reprezintă cele trei moduri de joc. La inițializarea acestei activități, în funcția *onCreate()* clientul încearcă o conexiune la server. Dacă aceasta conexiune reușește se va seta iconița de la statusul serverului în verde, altfel se va seta roșu.

[18] ***, Icoane disponibile pe http://www.iconarchive.com/

După ce utilizatorul introduce username-ul și parola, acestea se trimit către server folosind clasa *ConnManager* și apoi se citește răspunsul oferit de acesta. Dacă răspunsul este „*succes”* atunci, în funcție de butonul radio selectat, se creează o noua intenție care va porni activitatea specifică modului de joc selectat. De asemenea, utilizând metoda *putExtra* a unei intenții se va trimite către activitatea nou creată numele utilizatorului conectat.

Dacă jucătorul alege să se înregistreze în aplicație atunci se va crea o noua intenție care va deschide activitatea specifică înregistrării unui utilizator ce conține două câmpuri pentru a introduce numele său și parola. La apăsarea butonului de înregistrare se trimite mesaj către server cu cele două informații și dacă înregistrarea a avut loc cu succes atunci clientul va primi mesajul „*succes”*.

SinglePlayer

În cazul în care utilizatorul selectează modul de joc *SinglePlayer,* aplicația va porni activitatea *singleplayer_StartActivity.* În această activitate, se setează doar câte un *listener* pentru cele trei butoane: *Play, Score, Logout.*

La apăsarea butonului *Play* se creează o noua intenție care pornește activitatea selectmap_*Activity.* În cadrul acestei activități, se cer de la server toate hărțile disponibile pentru utilizatorul curent prin trimiterea mesajului „*getmap:username”*. Pentru a evita blocarea aplicației până la întoarcerea răspunsului cu toate hărțile de către server, această cerere se realizează în mod asincron utilizând paradigma *AsyncTask*. Aceasta are trei funcții: *onPreExecute* ce execută codul înainte de a face apelul asincron și unde vom afișa un obiect de tip *ProgressBar* ce indică încărcarea hărților, *doInBackground* ce trimite cererea către server și apoi rând pe rând citește toate hărțile și în cele din urmă funcția *onPostExecute* care se execută după terminarea cereri asincrone de la server. În această funcție vom construi în mod dinamic modul în care arată lista cu hărți primită de la server prin crearea de obiecte de tip *ImageView* pentru imagini și elemente *TextView* pentru numele traseului, cel mai bun rezultat și numărul de puncte de control.

La alegerea unui traseu, se creează o intenție nouă care va porni activitatea *maps_Activity.* Acestei activități i se transmit id-ul hărții, numele punctului de start, cel mai bun timp și numele utilizatorului. La pornire, activitatea încarcă, utilizând API-ul Google Maps, o hartă într-un element de tip *fragment*. De asemenea, pentru a încărca punctele de control pe parcursul jocului, se trimite o cerere asincronă către server iar utilizatorul nu va putea începe jocul până când toate punctele de control nu au fost încărcate. Când transferul este complet, folosind coordonatele primului punct, se

plasează pe hartă un obiect de tip *marker* unde jucătorul trebuie să ajungă. La apăsarea de către utilizator a butonului *Start the first puzzle,* folosind formula Haversine calculez distanța curentă a acestuia față de punctul de control, aspect evidențiat în *Listing nr. 11* și dacă această distanță este mai mică de zece metri atunci utilizatorul poate rezolva ghicitoarea. Dacă nu este în zona acestui punct, folosind API-ul Google Directions, trasez pe hartă o rută[19] din poziția curentă a utilizatorului până la poziția de start.

Listing nr. 11

```
    private double calculateDistance(LatLng firstPosition, LatLng
secondPosition){
        double R = 6371;
        double dLat = (firstPosition.latitude - secondPosition.latitude) *
Math.PI / 180;
        double dLon = (firstPosition.longitude - secondPosition.longitude) *
Math.PI / 180;
        double lat1 = firstPosition.latitude * Math.PI / 180;
        double lat2 = secondPosition.latitude * Math.PI / 180;

        double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
                Math.sin(dLon/2) * Math.sin(dLon/2) * Math.cos(lat1) *
Math.cos(lat2);
        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
        double d = R * c;
        double m = d * 0.621371;
        Log.d("distance is", String.valueOf(d));
        return d;
    }
```

Dacă jucătorul poate începe puzzle-ul, în partea de sus a ecranului va porni un cronometru ce va reține timpul de pe traseul curent. Acest cronometru este creat utilizând un obiect de tip *Chronometer*. Totodată, se va crea o nouă intenție care va porni activitatea *puzzle_Activity* utilizând modul *startActivityForResult.* Acest mod permite setarea, în activitatea nou creată, a unui rezultat, închiderea acestei activități și apoi preluarea rezultatului din prima activitate. Pentru transferul de

[19] Mathew G., Drawing driving route directions between two locations using Google Directions in Google Map Android API V2, Disponibil la: http://wptrafficanalyzer.in/blog/drawing-driving-route-directions-between-two-locations-using-google-directions-in-google-map-android-api-v2/

obiecte de tip *Marker* între cele doua activități am creat clasa *MapMarker* ce implementează interfața *Parceable* pentru a putea serializa datele din acest obiect. Astfel, în activitatea *puzzle_Activity* se preia obiectul de tip *MapMarker* aferent punctului de control la care utilizatorul trebuie să rezolve puzzle-ul și afișează pe ecran informațiile necesare precum întrebarea și opțional o imagine. Utilizatorul trebuie să răspundă la întrebare, răspunsul fiind verificat în obiectul primit la crearea activității. Dacă este un răspuns corect, se setează rezultatul activității pe *SUCCES* și apoi se închide activitatea prin apelarea metodei *finish().* În caz contrar se setează rezultatul pe *FAIL* aspect evidențiat in *Listing nr. 12.*

Listing nr. 12

```
public void checkAnswer(){
        EditText answerText = (EditText) findViewById(R.id.answerText);
        String answer = answerText.getText().toString();
        if(answer.equals(currentMarker.answer)){
            Intent returnIntent = new Intent();
            setResult(SUCCESS,returnIntent);
            finish();
        }
        else{
            Intent returnIntent = new Intent();
            setResult(FAIL,returnIntent);
            finish();
        }
    }
```

În activitatea *maps_Activity,* există funcția *onActivityResult* care preia răspunsul de la activitatea încheiată. Dacă a preluat răspunsul *SUCCES,* pe hartă va fi plasat un obiect de tip *marker* reprezentând noul punct de control și, folosind API-ul Google Directions, se trasează traseul către acest punct și se reiau pașii descriși mai sus.

După ce utilizatorul a trecut prin toate punctele de control, se preia timpul pentru acest traseu și se trimite către server în vederea actualizării dacă este un rezultat mai bun decât cel de până acum. De asemenea, jucătorul poate ieși de pe acest traseu sau poate reîncepe prin apăsarea butonului *Quit* respectiv *Restart map* care este accesibil pe tot parcursul jocului.

MultiPlayer

Dacă utilizatorul alege modul de joc *Multiplayer*, se va încărca activitatea *multiplayerstart_Activity* care va pune la dispoziția utilizatorului cele trei butoane: pentru găzduirea unui noi joc, pentru alăturarea la un joc nou creat și pentru deconectare. De asemenea, se va trimite un mesaj către server, în care se cere numărul de jucători conectați în acest moment la acest mod de joc. La primirea răspunsului, în partea de jos a ecranului va fi setat un mesaj corespunzător acestui număr.

În cazul în care utilizatorul alege să găzduiască un nou joc, se va deschide activitatea *selectMultiplayerMap_Activity* care, similar modului *SinglePlayer*, va încărca de la server, în mod asincron, hărțile disponibile. În schimb, la alegerea unei hărți, se va crea o intenție noua care deschide activitatea *lobby_Activity.* Această activitate reprezintă camera jocului tocmai creat, în care utilizatorul trebuie să aștepte conectarea altui jucător. La crearea acestei activități, dacă cel care a ajuns aici este cel care a creat jocul, atunci se va trimite către server mesajul *„creategame:mapid:username”* pentru a crea un nou joc iar butonul *Start* va deveni inactiv până la conectarea altui jucător. Dacă nu este cel care a creat jocul ci cel care a ales să se alăture unui joc deja creat și a ales acest joc din lista de jocuri încărcată în activitatea *onlineGamesSelect_Activity*, atunci se va trimite către server mesajul *joingame:gamename:username* pentru a se alătura la joc. Pentru a nu rămâne aplicația blocată, în această activitate, dacă utilizatorul este creatorul jocului, se va crea un nou fir de execuție care va aștepta mesajul „*updateplayers*” asta însemnând că un jucător s-a conectat la acest joc și butonul *Start* poate deveni activ, aspect redat în *Listing nr. 13*. Pe de altă parte, dacă utilizatorul este cel care s-a alăturat la joc, firul de execuție va aștepta mesajul „*startgame*” pentru a ști că a început jocul.

Listing nr. 13

```
            Thread t;
            StrictMode.ThreadPolicy policy = new
StrictMode.ThreadPolicy.Builder().permitAll().build();
            StrictMode.setThreadPolicy(policy);
            t = new Thread(new Runnable() {
                public void run() {
                    String receivedMessage =
ConnManager.getInstance().getMessage();
```

```
                    if(receivedMessage.equals("updateplayers")){
                        runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                mHost = false;
                                requestConnectedPlayersFromServer();
                                Button btn = (Button)
findViewById(R.id.startButtonMultiplayer);
                                btn.setEnabled(true);
                                mHost = true;
                            }
                        });
                    }
                }
            });
            t.start();
```

Dacă creatorul apasă butonul *Start,* aplicația va porni activitatea cu harta și va notifica serverul pentru ca cel care s-a alăturat să primească la rândul său notificarea și să pornească activitatea cu harta. Când ambii jucători au deschis activitatea *multiplayerMaps_Activity* se vor încărca, in mod similar modului *SinglePlayer*, toate punctele de control. Modul de joc este similar, singura diferență este cea că în partea de sus a ecranului este afișat numărul de puncte pe care îl mai are adversarul. Pentru a realiza actualizarea acestor puncte în timp real există un fir de execuție care așteaptă de la server mesajul *„updatemarkers:username:number”* care preia numărul de puncte rămase pentru adversar. Dacă acest număr este zero atunci înseamnă că adversarul a terminat traseul iar dacă are o valoare negativă (-1) atunci partenerul de joc a părăsit jocul. De asemenea, jucătorii au posibilitatea de a afla locația celuilalt participant prin apăsarea butonului *Reveal*, care trimite către server cererea de locație a adversarului și la primirea acesteia se va plasa pe hartă un obiect de tip *marker* de culoare albastră ce reprezintă poziția acestuia. Utilizatorii nu au voie să acceseze această opțiune decât de trei ori fiind indicat pe buton câte astfel de încercări mai au.

Dacă pe parcursul jocului, unul din cei doi apasă tasta *Back* a dispozitivului, se trimite către server mesajul „*updatemarker:username:number*” unde numărul este negativ, și totodată mesajul

„*deletegame:gamename”* deoarece nu mai este un joc valid. Jucătorul rămas în joc poate continua jocul ca și cum ar fi în modul *SinglePlayer*.

CreateMode

În cazul în care utilizatorul alege modul de joc *CreateMode*, se va crea o intenție nouă care pornește activitatea *createmode_startActivity,* care la rândul ei, prin apăsarea butonului *Create* va porni activitatea *startpoint_Activity*. În această activitate, trebuie ca utilizatorul să seteze punctul de start pentru noul traseu. Astfel, folosind o intenție cu ACTION_PICK, aplicația permite jucătorului să aleagă o imagine reprezentativă pe care o va afișa pe ecran după selectare. De asemenea, acesta trebuie să completeze numele punctului de start și numărul de puncte de control care vor fi pe hartă. La apăsarea butonului *Create,* clientul va trimite către server informațiile pe care utilizatorul le-a completat până în acest moment și va aștepta să primească id-ul noului traseu creat. După primirea acestuia se va crea o intenție care va deschide activitatea *createMap_Activity*. În această activitate, se va încărca o hartă care va avea un *listener* pe apăsarea lungă pe aceasta, aspect redat în *Listing nr. 14*, in interiorul căruia se va amplasa pe hartă un obiect de tip *marker*.

Listing nr. 14

```
    @Override
    public void onMapLongClick(LatLng latLng) {
        if(numberOfMarkers == 0){
            Toast.makeText(getBaseContext(),"You can not add more
markers!",Toast.LENGTH_SHORT).show();
        }
        else{
            mMap.addMarker(new MarkerOptions()
                    .position(latLng)
.draggable(true).icon(BitmapDescriptorFactory.fromResource(R.drawable.marker
)));
            markerClicked = false;
            numberOfMarkers--;
            Toast.makeText(this, "You have more "+numberOfMarkers+" markers
to put",
                    Toast.LENGTH_LONG).show();
        }
    }
```

De asemenea, obiectele de tip *marker* vor avea asociați *listeneri* pentru a putea fi mutați pe hartă la activarea opțiunii *Drag and Drop*, conform *Listing nr. 15* iar pentru meniul contextual ce conține opțiuni pentru a putea fi setat puzzle-ul asociat acestui punct sau ștergerea sa necesită apăsarea simplă pe acest punct.

Listing nr. 15

```
@Override
    public void onMarkerDragEnd(Marker marker) {
        Log.d("In drag end","Searching for marker");
        currentMarker = marker;
        for(int i = 0;i<mMarkersPair.size();i++){
            if(oldMarker.getPosition().longitude ==
mMarkersPair.get(i).marker.getPosition().longitude &&
oldMarker.getPosition().latitude ==
mMarkersPair.get(i).marker.getPosition().latitude){
                mMarkersPair.get(i).marker = currentMarker;
                mMarkersPair.get(i).mapMarker.x =
currentMarker.getPosition().latitude;
                mMarkersPair.get(i).mapMarker.y =
currentMarker.getPosition().longitude;
                Log.d("Found marker","Markers");
            }
        }
    }
```

La selectarea opțiunii de a seta un punct de control, se va deschide activitatea *puzzleCreate_Activity*, care, similar setării punctului de start, utilizatorul trebuie să scrie o întrebare, un răspuns și opțional poate selecta și o imagine. Totodată, acesta trebuie să selecteze și ordinea în care va apărea acest punct pe hartă. În funcție de această ordine, după ce toate punctele au fost setate pe hartă, se trimit pe server, rând pe rând, toate informațiile asociate punctelor de control. Apoi utilizatorul poate merge direct în modul *SinglePlayer* pentru a testa acest nou traseu creat.

Concluzii finale

Concluzii

Consider aplicația City Challenge fiind utilă utilizatorilor prin facilitățile pe care le oferă: crearea unui cadru de joc atractiv ce permite utilizatorului să își testeze și dezvolte atât abilitățile de orientare, cât și perspicacitatea, expunerea unui mod de joc în care utilizatorul poate confrunta alți participanți la joc și posibilitatea de a crea propriile trasee pe care să le prezinte altor utilizatori.

Pentru realizarea acestui proiect, m-am confruntat cu o serie de dificultăți cu privire la modul de joc *multiplayer* deoarece a fost nevoie de o sincronizare perfectă a firelor de execuție în vederea recepționării mesajelor corespunzătoare de către fiecare jucător conectat. Totodată, a fost necesară o coordonare ideală în ceea ce privește transferul de biți între cele două părți, client și server, pentru interpretarea potrivită a acestor biți. În final, am reușit să îmbin atât serverul cât și clientul într-un singur proiect stabil din punct de vedere funcțional și care oferă utilizatorului un mediu de joc antrenant și distractiv.

De asemenea, am utilizat cu succes API-ul Google Maps pentru a realiza interfața de joc la nivel de client și posibilitatea conectării mai multor astfel de clienți la server prin folosirea firelor de execuție care permit tratarea lor în mod simultan.

Direcții de viitor

Pentru viitor, aplicația ar putea fi îmbunătățită prin următoarele aspecte: adăugarea de moduri de joc noi, suportul pentru mai mult de doi utilizatori într-un joc online și eventual o hartă personalizată în care să fie mai bine delimitate zonele ce aparțin unui anumit traseu. De asemenea, harta ar putea include detalii suplimentare în ceea ce privește elementele din jur.

Bibliografie

1. Tanenbaum S. A., Computer Networks, Editura Wetherall, 2010
2. ***, Orientarea turistică – Fișă tehnică, Disponibil la: http://www.scoutpanaitescu.ro/campuri-activitati/Orientare_turistica.pdf
3. Jay P., September 2014 statistics about all app store, Disponibil la: http://www.whatech.com/mobile-apps/news/32469-september-2014-statistics-about-all-app-store
4. Stanciu L., Programare vizuală și modelare, Disponibil la: http://www.aut.upt.ro/~loredanau/teaching/PVM/Programare%20Vizuala%20si%20Modelare_intermediar.pdf
5. ***, Prezentare Android, Disponibil la: http://ocw.cs.pub.ro/courses/pdsd/labs/00
6. ***, Google Maps Android API ,Disponibil la: https://developers.google.com/maps/documentation/android/map
7. ***, Ingress: The game that reveals Google's secret war to control London, Disponibil la: http://www.theguardian.com/technology/2014/jun/04/ingress-the-game-that-reveals-googles-secret-war-to-control-london
8. Nuñal P., Best augmented reality (AR) games for Android, Disponibil la: http://www.androidauthority.com/best-augmented-reality-ar-games-for-android-93520/
9. Saket, 36 Awesome Augmented Reality Apps & Games for Android, Disponibil la: http://techsplurge.com/3214/mega-list-33-awesome-augmented-reality-apps-games-android/
10. ***, Top 10 Augmented Reality Games for Android / iOS platforms, Disponibil la: http://deepknowhow.com/2013/12/11/augmented-reality-games-android-iphone/
11. ***, Android, Disponibil la: http://cs.curs.pub.ro/wiki/si/lab/2013/android
12. Griffin I., Linux Network Programming, Part 1, Disponibil la: http://www.linuxjournal.com/article/2333
13. ***, Multithreading in C++, Disponibil la: https://www.tutorialcup.com/cplusplus/multithreading.htm
14. Mathew G., Drawing driving route directions between two locations using Google Directions in Google Map Android API V2, Disponibil la: http://wptrafficanalyzer.in/blog/drawing-driving-route-directions-between-two-locations-using-google-directions-in-google-map-android-api-v2/